DISTRIBUTED COMPUTING AND COMMUNICATION IN PEER-TO-PEER NETWORKS

by

Bradley Charles Goldsmith

Bachelor of Computer Science, University of Wollongong (1997)
Master of Computing, University of Tasmania (2004)

Submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

at the

UNIVERSITY OF TASMANIA

April 2010

Declaration of Originality

Chapter Three of this thesis contains some background material originally published in (Goldsmith 2006) as a technical report from the School of Computing, University of Tasmania.

Chapter Six of this thesis is an expansion of material originally published in (Goldsmith 2007) from the Proceedings of the 6th International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2007). Parts of Chapter One also contains extracts from this same paper.

Edited versions of the instructions given within Appendix C for reproducing the results in this thesis, and clustering with BOINC and Condor, have been self-published on the Internet as blog entries and now part referenced by the BOINC project as external documentation (BOINC 2007).

This thesis contains no material which has been accepted for a degree or diploma by the University of Tasmania or any other institution, except by way of background information which is duly acknowledged in the thesis. To the best of my knowledge and belief no material previously published or written by another person is included, expect where due acknowledgement is made in the text of the thesis.

Except where otherwise indicated, this is my own work.

Signed: .....

Dated: .....

Authority of Access

This thesis may be made available for loan and limited copying in accordance with the Australian Copyright Act 1968.

All trademarks mentioned in this thesis are the property of their respective owners.

Signed: .....

Dated: .....

Abstract

Traditional distributed computing systems are typically complex to implement and costly to maintain. Furthermore, little comparative work has been done to understand the performance and usability of these systems for their own sake as well of that of new approaches that eventuate. The work presented herein addresses both of these problems by describing the design and implementation of CompTorrent, a simple to implement and maintain decentralised peer-to-peer computing network, based on techniques used in other, non-computing peer-to-peer systems. This research also describes a new framework (WAN-DC) suitable for the comparison of wide area distributed computing systems. CompTorrent is compared with BOINC and Condor, two traditional distributed computing systems, using the WAN-DC framework on the same cluster hardware.

WAN-DC consists of a baseline to quantify the size of the cluster hardware followed by a set of well-known algorithms including calculations of the Mandelbrot set, a conversion of video formats (Transcode) and a ray-tracing of a benchmarking scene (POV-Ray). Other experiments include determining the systems underlying overhead with work units of no load (No Work), as well as work units of ranging sizes in order to measure where a system becomes acceptably efficient. This last test, One Second, is particularly useful when comparing different systems and approaches.

Results show that CompTorrent maintains a performance range between that of BOINC and Condor for all cluster node sizes for the POV-Ray experiment. Transcode shows CompTorrent is between or better than BOINC and Condor in 50% of cases and, whilst worse in the other half of experiments, it was only by approximately 15% in the worst case. Mandelbrot showed results between both distributed systems and, similarly to POV-Ray, No Work behaves either between or better than the results of both BOINC and Condor. These, and other, results described within, show that that a distributed processing system based on a decentralised, peer-to-peer network can provide similar results to distributed processing systems based on traditional client/server networking architectures.

This work demonstrates a convergence of peer-to-peer and distributed computing systems which while considered as certain as "death and taxes" (Foster & Iamnitchi 2003) has not, until now, been formally demonstrated in an academic setting for general purpose distributed computing where comparable systems exist based on a client-server approach. It is hoped that this work contributes to the further adoption of "Grassroots" distributed computing by bringing the ability to host and manage a distributed computing project to a much wider audience.

Acknowledgements

I would like to thank my supervisor, Dr. Daniel Rolf, for his supervision, patience and good humour during the course of this thesis. Thanks also to my associate supervisor, Dr. Waheed Hugrass, for his help during the latter stages of my candidature in making sure this thesis got out the door. I would also like to thank my fellow PhD students at the School of Computing in Launceston for their support and advice over the years. In particular, Alistair Atkinson and Michael Horton, with whom I most frequently whinged about our lot in life as PhD students. Michael also assisted me greatly by providing a very thorough proofread that was absolutely above and beyond the call of friendship. My partner, Emma Kirsopp, who whilst also a student herself for parts of this thesis, helped keep both the house and I together. I would also like to thank the anonymous examiners (who provided reports) that helped improve the quality of this dissertation.

This research was supported by a Tasmanian Graduate Research Scholarship.

This thesis document was produced with the following tools:

Text and layout were written with OpenOffice.org.
Graphs produced by GnuPlot and GraphViz.
Postscript and PDFs were produced with tools from GhostScript.

Table of Contents

1.0 Introduction....1
1.2 Motivation....1
1.3 Major Contributions....2
1.4 Structure of this Thesis....2
2.0 Distributed Computing, Peer-to-Peer Networks and Measuring the Performance of Computing Systems....4
2.1 Distributed Computing....4
2.1.1 On the Differences Between Distributed and Parallel Computing....4
2.1.2 Characteristics of Distributed Computing Systems: Terms, Actors and Roles.. .5
2.1.3 Characteristics of Parallel Architectures....8
2.1.3.1 Architecture....8
2.1.3.1.1 Flynn's Taxonomy....8
2.1.3.2 Characteristics of Parallel Problems....9
2.1.3.2.1 No Interprocess Communication (Independent or “Embarrassingly” parallel) and Parametric Problems....9
2.1.3.2.2 Inter-process Communication (Coarse and Fine grained parallel problems) 9
2.1.3.2.3 Ordered or Purely Sequential Problems....10
2.1.4 Synchronisation in Distributed Processing Systems....10
2.1.4.1 Message Passing....10
2.1.4.2 Shared Memory....10
2.1.4.3 Coordination Models....11
2.1.4.3.1 Tuple Spaces and Coordination Languages....11
2.1.4.3.2 Channel Based....11
2.1.5 Problem Solving in Distributed Systems....11
2.1.5.1 Data Decomposition....12
2.1.5.2 Task Decomposition....12
2.1.5.2.1 Recursive....12
2.1.5.2.2 Exploratory....12
2.1.5.2.3 Speculative....12
2.1.5.3 Hybrid Decomposition....13
2.1.6 Task Allocation in Distributed Systems....13
2.1.6.1 Tree....13
2.1.6.2 Crowd....13
2.1.6.3 Economic....13
2.1.6.4 Hybrid....13
2.1.7 Task Dependencies....14
2.1.7.1 Task Graph....14
2.1.7.2 Petri Nets....14
2.1.7.3 Heuristics....14
2.1.7.3.1 List Scheduling....14
2.1.7.3.2 Graph Decomposition....14
2.1.7.3.3 Critical Paths....15
2.1.7.4 Genetic Algorithms....15
2.1.8 Using Distributed Computing for Large Problem Solving....15
2.1.8.1 Advantages....15
2.1.8.2 Disadvantages....16
2.2 Peer-to-Peer networks....16
2.2.1 Characteristics of Peer-to-Peer Networks....16
2.2.1.1 Basic Peer-to-Peer Network Structures....17
2.2.1.2 Centralisation....17

2.2.1.3 Bootstrapping....17
2.2.2 Peer-to-Peer Network Architectures....18
2.2.2.1 Flooding Systems....18
2.2.2.2 Partially Centralised Systems....19
2.2.2.3 Hybrid Systems....19
2.2.2.4 Distributed Hash Tables....19
2.2.2.4.1 Trees....20
2.2.2.4.2 Skiplist....20
2.2.2.4.3 Cartesian Coordinate Space....20
2.2.2.4.4 DHT Summary....20
2.2.3 Advantages of a Peer-to-Peer Approach....20
2.2.4 Disadvantages of a Peer-to-Peer Approach....21
2.3 Benchmarking and Evaluation....21
2.3.1 Machine Performance....22
2.3.1.1 IPS and FLOPS....22
2.3.1.2 Benchmarks....22
2.3.1.2.1 Synthetic....23
2.3.1.2.2 Kernel....23
2.3.1.2.3 Component and I/O....23
2.3.1.2.4 Real Software....23
2.3.1.3 Simulation....23
2.3.2 Distributed and Parallel Systems....23
2.3.2.1 Speedup and Efficiency....24
2.3.2.2 Statistics and Leaderboards....24
2.3.3 Peer-to-Peer....24
2.3.3.1 Search Performance....24
2.3.3.2 Replication....24
2.3.3.3 Overlay Performance....24
2.4 Chapter Summary....25
3.0 Related Work....26
3.1 Peer-to-peer networks....26
3.1.1 Classification of Peer-to-peer Systems....26
3.1.2 Store and Forward....27
3.1.2.1 NNTP....27
3.1.2.2 SMTP....28
3.1.3 Centralised....28
3.1.3.1 Napster....28
3.1.4 Decentralised 28
3.1.4.1 Gnutella....28
3.1.4.2 Server Message Block....29
3.1.5 Structured 29
3.1.5.1 Distributed Hash Tables....29
3.1.5.1.1 Chord....29
3.1.5.1.2 Pastry....30
3.1.5.1.3 Content Addressable Network (CAN)....30
3.1.6 Unstructured 30
3.1.7 Hybrid....31
3.1.7.1 BitTorrent....31
3.2 Distributed Computing....32
3.2.1 Client / Server Systems....32
3.2.1.1 BOINC....32
3.2.1.2 Condor....32
3.2.2 Grid Computing....33
3.2.2.1 Globus Toolkit....33

3.2.2.2 Other Systems: P-Grid & X-Grid....33
3.2.2.3 Cloud Computing is not Grid Computing....34
3.3 Related Work Combining Peer-to-peer and Distributed Computing....34
3.3.1 Retrofitting BitTorrent into Distributed Computing Data Distribution....34
3.3.2 Peer-to-peer Distributed Computing....34
3.3.2.1 GPU....35
3.3.2.2 JXTA / JNGI....35
3.3.2.3 P-Grid....35
3.3.3 Botnets....35
3.4 Benchmarking and Evaluation....36
3.4.1 Whetstone and Dhrystone....36
3.4.2 Netperf....36
3.4.3 LINPACK and LAPACK....36
3.4.5 NAS Grid Benchmarks....37
3.4.6 Peer-to-peer Benchmark Work....37
3.5 Chapter Summary....37
4.0 WAN-DC: A New Framework for the Comparison of Wide Area Network Distributed Computing Systems....38
4.1 Motivation....38
4.2 Performance Metrics....39
4.3 Computing Benchmark....39
4.3.1 Baseline....40
4.3.1.1 LAPACK (synthetic)....40
4.3.1.2 Whetstone & Dhrystone (theoretical maximum)....40
4.3.1.3 Netperf / Netpipe (theoretical maximum)....41
4.3.2 Processing Intensive....41
4.3.2.1 POV-Ray Benchmark (application)....41
4.3.3 Mix....42
4.3.3.1 Transcode (application)....42
4.3.3.2 Mandelbrot (synthetic)....43
4.3.4 Responsiveness / Overhead....44
4.3.4.1 One second (synthetic)....44
4.3.4.2 Mean work unit (synthetic derived from application)....45
4.3.4.3 No work (theoretical maximum)....45
4.4 Qualitative Issues....46
4.4.1 General Approach & Design....46
4.4.1.1 Approach....46
4.4.1.2 Node Organisation (client / server, peer to peer)....47
4.4.1.3 Network Topology....47
4.4.1.4 Application....48
4.4.1.5 Requirements & Dedication....48
4.4.2 Features....49
4.4.2.1 Algorithmic Suitability....49
4.4.2.2 Standards Support....50
4.4.2.3 Hardware Support....50
4.4.2.4 Task & Resource Management....50
4.4.2.5 Robustness....51
4.4.2.6 Licensing....51
4.4.3 Usability....51
4.4.3.1 Hosting....51
4.4.3.2 Joining....52
4.4.3.3 Coding....52
4.4.3.4 Support....52
4.4.4 Incentives....52

4.5 Discussion of the WAN-DC benchmark....53
4.6 Chapter Summary....53
5.0 A Comparative Evaluation of Condor and BOINC Using the WAN-DC Benchmark. .54
5.1 Test Platforms....54
5.2 Test Environment....54
5.3 Test Datasets....54
5.4 Results....55
5.4.1 Baseline....55
5.4.1.1 LAPACK....55
5.4.1.2 Whetstone & Dhrystone....55
5.4.1.3 NetPerf....56
5.4.2 Condor....57
5.4.2.1 POV-Ray....57
5.4.2.2 Transcode....57
5.4.2.3 Mandelbrot....57
5.4.2.4 No Work....58
5.4.2.5 One Second....58
5.4.2.6 Mean Work Unit....61
5.4.3 BOINC....62
5.4.3.1 POV-Ray....62
5.4.3.2 Transcode....62
5.4.3.3 Mandelbrot....62
5.4.3.4 No Work....63
5.4.3.5 One Second....63
5.4.3.6 Mean Work Unit....66
5.4.4 Qualitative Evaluation....66
5.5 Discussion of Performance Results on BOINC and Condor....68
5.5.1 POV-Ray....68
5.5.2 Transcode....69
5.5.3 No Work....69
5.5.4 Mandelbrot....69
5.5.5 One Second....69
5.5.6 Mean Work Unit....69
5.6 Chapter Summary....69
6.0 CompTorrent 71
6.1 Introduction....71
6.2 Technical Overview....73
6.2.1 Metadata File....73
6.2.2 Tracker....74
6.2.3 Node....75
6.2.4 Security....76
6.3 Using CompTorrent....77
6.3.1 A Suitable Algorithm (and data!)....77
6.3.2 Locating a Tracker....77
6.3.3 Creating the Metadata File....77
6.3.4 Planting the Seed....78
6.3.5 Seeing the Results....79
6.3.5.1 The home page....79
6.3.5.2 Node & Work Unit List....79
6.3.5.3 Overlay Graph....80
6.3.5.4 Connection Graph....80
6.3.5.5 Last Run Times....81
6.4 Chapter Summary....81
7.0 CompTorrent Implementation....82

7.1 Major Components....82
7.1.1 Tracker....82
7.1.1.1 Communication Protocol with Nodes....82
7.1.1.2 Discussion of Tracker Design....88
7.1.2 CompTorrent Application....89
7.1.2.1 Major Objects....89
7.1.2.1.1 Main Loop....89
7.1.2.1.2 Controller....90
7.1.2.1.3 CompTorrentPeer....91
7.1.2.1.4 TrackerParser....92
7.1.2.1.5 CompTorrentParser....92
7.1.2.1.6 Listener....93
7.1.2.1.7 Router....93
7.1.2.1.8 Processor....94
7.1.2.2 Communication Protocol Between Nodes....94
7.1.2.3 Network Overlay & Topology....96
7.1.2.4 Security....97
7.2 The CompTorrent Metadata File....97
7.3 CompTorrent in Action....99
7.3.1 A Seed Node by Itself....99
7.3.2 A Node Joining Without a Route ID....100
7.3.3 A Node Joining a Node....100
7.3.4 Requesting & Completing a Work Unit....100
7.3.5 Verification of the Work Unit....101
7.3.6 Assembling the Results....101
7.4 Summary....101
8.0 CompTorrent Evaluation....102
8.1 Performance Results....102
8.1.1 POV-Ray....102
8.1.2 Transcode....102
8.1.3 Mandelbrot....103
8.1.4 No Work....103
8.1.5 One Second....104
8.1.6 Mean Work Unit....106
8.2 Qualitative Results....108
8.3 Discussion of Results Compared with BOINC & Condor....109
8.4 Chapter Summary....111
9.0 Conclusions & Further Work....113
9.1 Conclusions....113
9.1.1 Distributed Computing with Peer-to-Peer Computing....113
9.1.1.1 Comparative Performance and Scalability....113
9.1.2 Ease of Use....113
9.1.3 New Incentives, Network Time, Processing Time and Machine Dedication. .114
9.2 Further Work....114
9.2.1 WAN-DC....114
9.2.1.1 Performance Metrics....115
9.2.1.2 Qualitative Metrics....115
9.2.1.3 More Systems & Underlying Network Conditions....115
9.2.2 CompTorrent....116
9.2.2.1 Protocol and Routing Optimisation....116
9.2.2.2 Interprocess Communication....116
9.2.2.3 Optimisation of File Transfer....116
9.2.2.4 Trackers....117
9.2.3 Botnets....117

9.3 Some Personal Concluding Remarks on Peer-to-Peer as a Controversial Research Topic....117
10.0 References....119
Appendix A....127
Building Source Code....127
Tracker Database Schema....127
Tracker Database Schema Diagram....129
UML Class Diagrams....129
Appendix B....133
Amdahl's Law....133
Gustafson-Barsis' Law....133
Appendix C....134
Experimental Setup Instructions....134
BOINC....134
Server....134
Client Side....136
Condor....142
Master Machine....142
Client Machine....143
Appendix D....145
WAN-DC Extras....145
Sequential read/write with underlying network changes (synthetic)....145
Input/Output Intensive....145
Sequential read/write (synthetic / application)....146
BitTorrent (theoretical maximum)....146
Failure and Malfeasance....147
Erroneous Results (synthetic)....147
First work unit (synthetic derived from application)....148
Appendix E....149
Data Sets....149
Transcode....149
POV-Ray....150
Mandelbrot....150
One Second....151
Appendix F....152
Experimental Results....152

List of Figures

Figure 1: Ethernet throughput of the cluster network....56
Figure 2: The round trip time of the cluster network with packets of increasing size....56
Figure 3: Condor One Second graph for 1 machine....59
Figure 4: Condor One Second graph for 2 machines....59
Figure 5: Condor One Second graph for 4 machines....59
Figure 6: Condor One Second graph for 8 machines....60
Figure 7: Condor One Second graph for 16 machines....60
Figure 8: Condor Mean One Second results....60
Figure 9: BOINC One Second graph for 1 machine....63
Figure 10: BOINC One Second graph for 2 machines....64
Figure 11: BOINC One Second graph for 4 machines....64
Figure 12: BOINC One Second graph for 8 machines....64
Figure 13: BOINC One Second graph for 16 machines....65
Figure 14: BOINC Mean One Second Results....65
Figure 15: A new job is started by the creation and publication of a metadata file....72
Figure 16: The major steps in joining a CompTorrent swarm....73
Figure 17: Three nodes is a simple network interacting with each other and the tracker....76
Figure 18: The tracker WWW interface home page....79
Figure 19: The tracker Node & Work list....79
Figure 20: The overlay graph....80
Figure 21: The IP connection graph....80
Figure 22: The tracker page of last run times and other statistical information....81
Figure 23: Collaboration graph for the sample implementation of CompTorrent....89
Figure 24: CompTorrent One Second graph for 2 machines....104
Figure 25: CompTorrent One Second graph for 4 machines....104
Figure 26: CompTorrent One Second graph for 8 Machines....105
Figure 27: CompTorrent One Second graph for 16 machines....105
Figure 28: CompTorrent Mean One Second results....106

Index of Tables

Table 1: Flynn's Taxonomy shown as a matrix....8
Table 2: A new classification scheme by peer-to-peer approach along with its generative classification equivalent and examples....27
Table 3: The LAPACK experiment in detail in the WAN-DC benchmark....40
Table 4: The Whetstone and Dhrystone experiment in detail in the WAN-DC benchmark. 41
Table 5: The Netperf experiment in detail in the WAN-DC benchmark....41
Table 6: The POV-Ray experiment in detail in the WAN-DC benchmark....42
Table 7: The Transcode experiment in detail in the WAN-DC benchmark....42
Table 8: The Mandelbrot experiment in detail in the WAN-DC benchmark....44
Table 9: The One Second experiment in detail in the WAN-DC benchmark....45
Table 10: The Mean Work Unit experiment in detail in the WAN-DC benchmark....45
Table 11: The No Work experiment in detail in the WAN-DC benchmark....46
Table 12: The qualitative approach category in the WAN-DC benchmark....47
Table 13: The qualitative node organisation category in the WAN-DC benchmark....47
Table 14: The qualitative network topology category in the WAN-DC benchmark....48
Table 15: The qualitative application category in the WAN-DC benchmark....48
Table 16: The qualitative requirements category in the WAN-DC benchmark....49
Table 17: The qualitative algorithm support category in the WAN-DC benchmark....49
Table 18: The qualitative task ordering category in the WAN-DC benchmark....49
Table 19: The qualitative communication standards category in the WAN-DC benchmark. 49
Table 20: The qualitative hardware support category in the WAN-DC benchmark....50
Table 21: The qualitative job allocation category in the WAN-DC benchmark....50
Table 22: The qualitative robustness category in the WAN-DC benchmark....51
Table 23: The qualitative licensing category in the WAN-DC benchmark....51
Table 24: The qualitative usability category in the WAN-DC benchmark....51
Table 25: The qualitative hosting category in the WAN-DC benchmark....52
Table 26: The qualitative joining category in the WAN-DC benchmark....52
Table 27: The qualitative coding category in the WAN-DC benchmark....52
Table 28: The qualitative support category in the WAN-DC benchmark....52
Table 29: The qualitative incentives category in the WAN-DC benchmark....53
Table 30: Individual cluster machine performance for LAPACK....55
Table 31: Individual cluster machine performance for Whetstone and Dhrystone....55
Table 32: Condor results for the POV-Ray experiment....57
Table 33: Condor results for the Transcode experiment....57
Table 34: Condor results for the Mandelbrot experiment....58
Table 35: Condor results for the No Work experiment....58
Table 36: Condor results for the Mean Work Unit experiment....61
Table 37: BOINC results for the POV-Ray experiment....62
Table 38: BOINC results for the Transcode experiment....62
Table 39: BOINC results for the Mandelbrot experiment....63
Table 40: BOINC results for the No Work experiment....63
Table 41: BOINC results for the Mean Work Unit experiment....66
Table 42: WAN-DC qualitative results for both BOINC and Condor....68
Table 43: The major web based components of the CompTorrent Tracker....88
Table 44: The message schema of the CompTorrent protocol....95
Table 45: CompTorrent results for the POV-Ray experiment....102
Table 46: CompTorrent results for the Transcode experiment....103
Table 47: CompTorrent results for the Mandelbrot experiment....103

Table 48: CompTorrent results for the No Work experiment.....103
Table 49: Mean Work Unit results for CompTorrent.....107
Table 50: WAN-DC qualitative results for CompTorrent.....109
Table 51: POV-Ray speedup results for the three systems.....109
Table 52: Transcode speedup results for the three systems.....110
Table 53: Mandelbrot results for the three systems.....110
Table 54: No Work results for the three systems.....110
Table 55: Mean work unit (Transcode) results for the three systems.....111
Table 56: Mean work unit (Mandelbrot) results for the three systems.....111
Table 57: Mean work unit (POV-Ray) results for the three systems.....111
Table 58: Mean work unit (No Work) results for the three systems.....111

Chapter One

"Dissent is the native activity of the scientist, and it has got him into a good deal of trouble in the last years. But if that is cut off, what is left will not be a scientist. And I doubt whether it will be a man."

-- The late, great Dr. Jacob Bronowski.

1.0 Introduction

This thesis shows that a distributed processing system based on a decentralised, peer-to-peer network can provide similar results to distributed processing systems based on traditional client/server networking architectures. It describes the implementation of a new distributed processing platform (called CompTorrent) that achieves this level of result performance using a peer-to-peer network. CompTorrent is the first working system that demonstrates a convergence between hybrid peer-to-peer systems and distributed computing for general purpose distributed processing applications.

CompTorrent brings several new ideas and new possibilities to wide area distributed computing.

- Using a decentralised, ad hoc swarm as a processing resource for distributed computing
- Distribution of source and computed data sets concurrently with computation to enable participants to acquire both of the full data sets
- Using the acquisition of original and computed data sets as an incentive to join the computing swarm
- Minimising infrastructure requirements, in both network connectivity and server hardware, to a level approaching nothing for hosting a distributed computing project

The result is a similar level of performance to traditional distributed processing implementations with the added benefits of a decentralised, peer-to-peer architecture. This provides access to distributed processing resources that may not have been available to start up projects previously without the investment of time and resources to maintain a central server pool as required in a traditional client/server approach.

1.2 Motivation

Distributed computing has had several high profile successes from the 1990s onwards. Folding@Home (Larson et al 2003), since 2000, has been working on computing simulations for molecular dynamics to better understand certain diseases. Distributed.net (Haynes 1998) has spent the last 10 years answering challenges by RSA Security to encourage research into computational number theory. SETI@Home has had over 5.2 million participants processing data from the Arecibo radio telescope making it the largest distributed computing project over a wide area network to date (Anderson et al 2002). These projects, and others like them, are interesting, worthwhile and largely centralized in their control. This centralised approach has led to many processing participants, but relatively few distributed computing projects. This difference becomes particularly apparent when wide area

network distributed computing is compared to other distributed applications such as file sharing or instant messaging. There are potentially many more applications for raw computing power that could be realised should the overhead of centralised control be removed.

Another motivation has been the limited application of peer to peer techniques themselves and their individual suitability to some parts of distributed computing. File sharing has undoubtedly been the biggest, and most controversial, application of this technique and a large part of distributed computing is the distribution of data. Instant messaging and telephony has been successful with a peer-to-peer approach and the coordination of computing jobs has been a big part of wide area network distributed computing that has been managed centrally. Wide area network distributed computing requires significant computing resources just to centrally manage the computation. Peer-to-peer networking has produced large amounts of storage and computing power with minimal or no cost in terms of central resources. Denial of service attack, and other hacking attempts, have occurred several times on large centralised services (Vixie et al 2002) (ICANN 2007), including instances of public distributed computing projects (Kotadia 2004). Peer-to-peer systems have proven extremely resilient in the face of a variety of disruptive attacks.

This research has taken the first steps in showing that decentralised peer-to-peer distributed computing is feasible and can provide the tools to bring distributed computing, as a general purpose computing tool, to a much larger audience.

1.3 Major Contributions

In addition to showing that a decentralised, peer-to-peer network can provide similar results for distributed computing applications compared to existing client/server architectures the following research contributions have been made:

- Applying the metadata concept to distributed computing
- A framework for the comparison of distributed computing architectures that consists of performance measurement and selection criteria
- A comparison of two existing distributed computing systems (BOINC and Condor) on the same hardware and network platform
- Having the sharing and dissemination of the original and computed data sets as incentives for participating in an open distributed computing project
- Sharing the original dataset, computed dataset and execution load concurrently
- The comparison of computing performance of this newly developed system with two other established distributed computing platforms. The experimentation utilised the same hardware and network to provide a fair comparison without needing to rely on simulation

1.4 Structure of this Thesis

Chapter two provides the necessary background for the reader by giving a description of distributed computing, peer-to-peer networks and the measurement and benchmarking of distributed systems. Chapter three surveys a range of large and small projects that bear relation to the system described in this thesis in chapters six, seven and eight. Chapter four provides criteria for the evaluation of distributed computing systems by first describing a taxonomy of computing distributed systems and then a range of algorithms and techniques to enable a comparative analysis of existing and emerging systems. Chapter five presents an evaluation of the BOINC

and Condor distributed computing systems using the criteria presented in Chapter four. Chapters six and seven describe the design and implementation of the CompTorrent distributed computing system. The design principles of CompTorrent are explained followed by the actual implementation details and some discussion of decisions made. Chapter eight provides a detailed evaluation of the CompTorrent protocol using the evaluation framework presented in chapter four. Chapter nine provides a discussion of further work that has been identified during this research and then concludes.

Chapter Two

2.0 Distributed Computing, Peer-to-Peer Networks and Measuring the Performance of Computing Systems

This chapter presents an overview of the separate fields of distributed computing, peer-to-peer networking and measuring the performance of computing systems. It intends to give the reader the necessary technical background, of all three fields, to fully understand the rest of this thesis. This chapter concentrates on theoretical background and underlying concepts with examples only where necessary for explanation. The next chapter, chapter three, is where examples of these technologies are described.

2.1 Distributed Computing

Distributed computing refers to two or more computers networked together sharing the same computing work. The aim is that by sharing the job between multiple computers, the computing work will be completed more quickly than on one machine alone.

2.1.1 On the Differences Between Distributed and Parallel Computing

Whilst there are many similarities between the goals, research problems and operation of Parallel Computing and Distributed Computing, there are several subtle differences. These two different terms are often, and incorrectly, used synonymously. This confusion is further compounded by the fact that these two areas of research are constantly undergoing significant change. Leopold (2001) provides perhaps the clearest summary on the difference between parallel and distributed computing:

> "Parallel computing splits an application up into tasks that are executed *at the same* time, whereas distributed computing splits an application up into tasks that are executed *at different locations*, using *different resources*."

Parallel computing typically refers to multiple processing elements existing within one machine with each processor being dedicated to the overall system at the same time. Distributed computing refers to a group of separate machines that each contributes processing cycles to the overall system, over a network, over time.

Volunteer distributed computing was inspired by parallel computing systems yet driven by a lower cost solution of using cheaper individual machines interconnected by a local or wide-area network rather than specialist, expensive parallel hardware required to coordinate many processors within the same machine. Faster and cheaper networks have seen distributed systems emerge in the 1970s and 1980s, grow significantly through the 1990s to now being relatively common in research institutions in the 2000s. Distributed computing is, at time of writing,

undergoing a convergence due to improvements in personal computing hardware and wide area network performance.

This thesis is at all times concerned with distributed computing systems however several of the common characteristics will be explained. When the reference is to a parallel description, its is discussing the terms which are common to both parallel and distributed systems and algorithms. Otherwise, a distributed specific description, will refer to distributed computing systems and algorithms.

2.1.2 Characteristics of Distributed Computing Systems: Terms, Actors and Roles.

There are several elementary distributed computing terms and ideas which will first be defined for volunteer distributed computing. These notions are the computing job itself, the processing nodes and tasks being completed.

A *job* relates to the overall computing work that needs to be done in order to solve the problem at hand. There are several different categories of computing jobs based on the underlying nature of the computation or algorithm itself. Some jobs are very parallel in nature. These can be easily split into smaller parts that each processor can work on independently. These are known as *independently* or *embarrassingly parallel* problems. Some jobs only have a relatively small parallel component and due to this much communication needs to take place between the different entities working on different parts of the problem. This describes a problem which requires much *synchronisation* between the different parts of the problem being processed. The term used here to describe how well a job can be parallelised is *granularity*. A *coarse-grained* algorithm allows the problem to be split into sizeable chunks allowing more useful processing time compared to the time spent with communication over a network. A *fine-grained* problem describes the opposite; the job can only be processed for a relatively short period of time before communication is required.

A *node* is an entity on the network that is able to perform computing tasks. In a conventional parallel system this would refer to a physical processor unit within the computer system. In the traditional distributed computing scenario this is more likely to refer to a computer that is a part of the network. The modern practical reality tends to be a combination of the two. Modern systems can have multiple CPUs per machine and two and four processor systems are not unheard of in high-end personal computers. Since mid 2006, “dual core” or “multi core” processors have become the norm taking over from multiple discrete CPUs in a system. These are multiple independent CPUs that are housed in the one chip package; often on the same integrated circuit. This distinction is important as a discrete node on a distributed processing network might well internally be a parallel processor as well.

A *task* is a logically discrete part of the overall processing job. Each task is distributed amongst different machines or processors on the network to work on different parts of the overall computation job at the same time. It should be the overall aim of each distributed processing system to complete the computation work as efficiently as possible. In the literature, jobs are sometimes referred to as tasks and tasks referred to as subtasks.

Now that there is an understanding of the network of nodes that are each working on a task in order to work towards completing the overall job, there needs to be a way to coordinate this overall process. To do this one of the different types of ways that nodes can be organised and coordinated, and the tasks themselves

managed, must be used. This will now be considered.

There is often ordering associated with the computation of tasks in an overall job. The ordering may be sequential where the output from one particular task may provide input for another task. Or perhaps a group of tasks sharing the same data set may have shared data boundaries (or edges) where communication between tasks will need to occur for the job to be processed correctly. Also, whilst maintaining communication, these tasks need to be as evenly distributed as possible amongst nodes (or directed towards some and away from others) in order to maintain a balance. Maintaining synchronisation between nodes doing tasks in a dynamic environment is difficult as deadlocks[1] and race conditions[2] between different tasks and resources can occur. This calls for the need of *synchronisation* of tasks and is a feature of many distributed systems. Synchronisation can be centralised, with one node or agent in control of managing tasks, or decentralised with several or all nodes managing a part of the overall synchronisation control or a combination of both.

Of course, managing these nodes and allocating tasks incurs a communication overhead beyond what would occur if the tasks were being executed sequentially, that is one after the other, on a single machine or processor. There are three broad categories of overhead here. The first two, bandwidth and latency, are mostly influenced by the network underlying the distributed computing system. The third is response time which is the administrative time taken for the system to respond.

Bandwidth refers to how much data can be passed over a communication channel in a finite period of time. This is a term that has been loaned from the field of radio communications where there it refers to the difference between the upper and lower frequencies of a communications channel. In computer networking it refers to the data rate, or the amount of data that pass through a communication channel over a given period of time[3]. In distributed computing, especially in a fine grained problem, the amount of communication that needs to take place between different nodes is crucial in the overall efficiency of the system. The bandwidth of the network often becomes far more critical than the speed of the processing nodes themselves.

Latency refers to the time between an action being initiated and the action actually having some effect. In terms of the underlying network, latency can be the time between the data being sent and the data actually being received. In terms of a task, one measure of latency is the time between a task being allocated to a node and the node actually beginning the processing task. So again there is one term referring to two different things in the same area. Network latency is a serious issue along with bandwidth in determining the overall effectiveness of a distributed processing system. In this thesis latency issues in the underlying network will be referred to as *network latency* and delays in the processing job being allocated and returned as *response time*. Response time and network latency are often bundled together by the term *parallel overhead*. The time taken to create each task, start and stop threads of execution, wait for synchronisation cues and the network communication are all examples of issues that can significantly count towards overall execution time when compared to running the job on a single processor.

---

1 A deadlock occurs where two or more tasks are waiting for the other to finish first so neither ever does.

2 A race condition describes a problem where the output result from a task changes based on the sequence or timing of other events.

3 With the advent of wireless computer networking, the borrowing of this term has become a bad idea. It is now possible to use the word bandwidth twice in a sentence with each occurrence referring to a different thing.

Once the network of nodes is organised and the job is on its way to being computed and an understanding that there are delays and performance issues, there should exist a system that can be measured or at least a system where measurements would be a good thing to have. In distributed computing there are two elementary measures called *speedup* and *efficiency* that allows the practitioner to compare how the system is working against theoretical ideals.

Speedup is the ratio which describes how many times faster a job is running on multiple processors rather than a single processor. It is derived by taking the running time of a job running on one processor and dividing it by the running time of the same job running on two or more processors. This gives us a number that is a measure of how many processors the system is representing. Rarely is it the case that two processors will process the job exactly twice as fast as 1 processor due to the overheads already discussed that are inherent to distributed processing. This ideal measure is known as *linear speedup* and is often included on speedup graphs as a baseline for which the actual speedup measurements per number of processing elements is compared. Where speedup is on the *y* axis and and number of processors on the *x* axis, linear speedup will draw a straight line at 45˚ between the axes. The reality of course is that speedup will typically lag somewhat beneath the ideal and often the difference will increase as the number of processors does.

Occasionally, *super-linear speedup* can be observed. This is where for example 2 processors working on the job will show more than 2 times speedup. This is usually explained by caching, where the job can run faster overall due to the accumulation of cache sizes of all the involved processors. So if only half of the dataset fits within the cache on one node, split between two equal nodes the whole dataset could be stored in faster cache memory, rather than half cache and half conventional memory. This could potentially produce a speedup greater than the sum of the processors involved - for a few multiples of nodes anyway depending on data and cache size.

In parallel and distributed computing terms, *efficiency* is defined as the amount of speedup divided by the number of processors being utilised. That is, speedup per processor. This is typically a real number between 0 and 1.

These measures of speedup and efficiency can quickly allow us to determine the *scalability* of the system or how well will it continue to perform as more nodes and tasks are added. A naïve approach to measuring a distributed system might lead one to think that the more processing nodes that can be added to a system the better. Amdahl's Law (Amdahl 1967) shows us that there will always be an upper bound to scalability in a distributed computing system so much so that eventually adding more processors to a job can make the overall execution time longer as the overhead of communication and management increases.

Amdahl's Law generally states that every computing job contains parts that are not able to be executed purely in parallel. That means that no matter how many processing nodes are available to the problem, the sequential only part of the job will provide an upper limit to the overall speedup achieved[4]. This is a limitation until one considers that a problem that is sufficiently large enough to be distributed will often lend itself to parallelisation through the nature of its own size. Therefore the serial limitation of the job does not have so large an effect and that many, many nodes can be involved without speedup decreasing. This effect is described by Gustafson-Barsis' Law[5] and simply states that any sufficiently large problem can be efficiently

4 A Generalised formal definition for Amdahl's Law is given in Appendix B.

5 A Generalised formal definition for Gustafson-Barsis' Law is given in Appendix B.

parallelised.

This chapter will now move on to describe the different classifications of architectures and the different categories of problems that can be solved with distributed computing.

2.1.3 Characteristics of Parallel Architectures

There are many different categorisations of parallel machines and problems. Considered here will be two broad categorisations, the classic classification of Flynn's taxonomy as well as common classifications used in grid and cluster literature.

2.1.3.1 Architecture

Whilst there are many taxonomies in the literature for describing computer architectures, Flynn's Taxonomy has endured as the most general and useful for categorising existing and emerging architectures.

2.1.3.1.1 Flynn's Taxonomy

Flynn's taxonomy (Flynn 1972) is a classification of computer architectures that is also regularly used to classify algorithms as well.

	Single Instruction	**Multiple Instruction**
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

Table 1: Flynn's Taxonomy shown as a matrix.

Single Instruction, Single Data refers to a architecture that consists of a single processing unit operating on a single stream of data without any parallelism whatsoever. This is analogous to an early personal computer or a classic Von Neumann architecture.

Single Instruction, Multiple Data is an architecture where a single stream of instructions are applied to multiple streams of data simultaneously. Examples would include a processor optimised for array based operations or a graphical processor.

Multiple Instruction, Single Data is an architecture where multiple streams of instructions are applied to a single stream of data. Of the taxonomy, this is the rarest seen applied, as only the most fault tolerant designs require separately developed systems operating to provide a consensus result. Examples are found in aerospace applications such as flight control systems.

Multiple Instruction, Multiple Data is an architecture of multiple operating processors working on multiple streams of data. Multi threaded programming is often MIMD and distributed systems, where asynchronous operations by multiple processors on separate data, are a clear example.

2.1.3.2 Characteristics of Parallel Problems

In distributed computing terms, algorithms can be placed into three general categories: No interprocess communication, interprocess communication and sequential problems.

2.1.3.2.1 No Interprocess Communication (Independent or "Embarrassingly" parallel) and Parametric Problems

As already alluded to in 2.1.2, an independently parallel or embarrassingly parallel problem is one that can be completely computed with no communication between processing nodes whatsoever during the computing process. A purely embarrassingly parallel problem is one that includes no post processing of results.

Common examples of this class of algorithm would include a brute force search of a keyspace, calculating a fractal such as the Mandelbrot set or ray-tracing a graphical scene. Many classic wide area network distributed computing systems concentrate solely on this class of algorithm as the lack of communication allows for greater scalability.

No interprocess communication problems can be broken further down into two sub categories: parametric and data parallel problems. A parametric problem is an embarrassingly parallel problem that needs to be calculated multiple times with different parameters. For example a physical simulation may require the same calculations applied to the same data with different starting parameters as a problem set. After the multiple calculations have occurred, post processing may be required in order to chose the best fitting result or produce an aggregation of results. A data parallel problem is one where the data are evenly divided between processing nodes before the same algorithm is applied. The difference to plain embarrassingly parallel is that the computation amount may vary depending on which data are allocated. After computation has finished post processing is usually required to combine the result set.

2.1.3.2.2 Inter-process Communication (Coarse and Fine grained parallel problems)

This category of problems consists of algorithms that, when divided amongst processors, requires communication between them in order to correctly compute the given data. Depending on the nature of the algorithm, some require much communication (fine grained) or communicate relatively infrequently (coarse grained). Still, in the case of distributed computing applications, any communication between processors is extremely expensive in terms of slowing down the overall computation.

The amount of data to be communicated transcends the coarse/fine distinction – some problems may have small amounts of data shared between processors but very frequently. Others may have large amounts of data infrequently transmitted. A heat transfer computation for example will often have data shared between nodes at the boundaries of each data set only.

It is also noteworthy that some otherwise interprocess communication problems can be solved in an embarrassingly parallel way by recalculating some parts of the problem that would otherwise need to be shared.

2.1.3.2.3 Ordered or Purely Sequential Problems

A sequential problem is one where each task in the overall job must be calculated one after the other in a strict order. Typically, data must also be handled in a strict order as well. Problems with a high level of data dependency, when they occur, can render a problem unsuitable for parallelisation. However, as the size of the problem or the data set increases, generally there are opportunities for parallelisation. In practise, relatively few purely sequential problems exist. However an example of one would be the calculation of Fibonacci numbers. The dependence on the two preceding results removes the ability for the parallelism to increase enough (and still produce a Fibonacci sequence) for Gustafson-Barsis' Law to apply.

2.1.4 Synchronisation in Distributed Processing Systems

There are several models for communication between processing units in distributed computing or parallel computing systems. This inter-process communication is required for the management of the computing network as well as the synchronisation of computation as required by the algorithm being computed. There are many implementations of interprocess communication models in distributed computing, however they broadly fit into three underlying categories. These are *message passing*, *shared memory* or *coordination models*.

2.1.4.1 Message Passing

Message passing models rely on messages being sent and received between nodes working on a task over a network or bus. This is the only mode of communication so there is no shared memory between nodes. Message passing relies on the programmer to deal with all facets of the data distribution, scheduling and algorithm management. It requires a very low-level understanding of the message passing system being applied and multi-process programming beyond the domain knowledge required for the problem actually being solved itself.

The most common implementation of message passing is the Message Passing Interface (MPI). MPI is a language-independent communication protocol used for task synchronisation and is considered the de facto standard for distributed processing by message passing.

See Snir et al (1996) for a definitive guide to MPI.

2.1.4.2 Shared Memory

Shared memory models rely on all processing nodes having access to a shared, distributed memory store which is accessible to nodes working on a task. From a traditional point of view, shared memory programming can best be described as separate processing threads with a single process. Each thread executes independently of one another yet has a shared memory space that is visible, and editable, by all threads within the process. This greatly reduces the burden on the programmer, when compared to message passing, as individual threads need not know about other processing threads in order to carry out their task.

In a distributed environment, a distributed shared memory is maintained over physically separate machines, and hence separate memory spaces, yet access is typically available through an abstraction that looks like local memory. This allows for ease of programming at the significant cost of performance when access to memory is required from a non-local memory. Again, to draw from a traditional

SISD architecture, distributed shared memory is analogous to virtual memory in an operating system. Memory fetches in physical ram (local storage) is much faster than a fetch that requires a page to be brought from the disk (remote machine).

There are a variety of distributed shared memory projects under way with OpenMP as the defacto standard. See Protić et al for a comprehensive review of foundation concepts (Protić 1997) and the OpenMP official homepage (OpenMP 2009) for OpenMP resources.

2.1.4.3 Coordination Models

Coordination Models are a type of either shared memory or message passing that differ enough from their underlying model to be considered a synchronisation model in their own right. Coordination models differ themselves, from shared memory or message passing, through the computation and communication aspects of completing a task being completely separate from one another. This contrasts with the shared memory or message passing approaches where the communication is mixed with the computation from the developer's perspective.

2.1.4.3.1 Tuple Spaces and Coordination Languages

Tuple spaces allow the coordination of a computation job through a distributed, associative shared memory which consists of objects called tuples. A tuple is an ordered list of values that is referenced by its content rather than its address. Any process can access any tuple in the tuple space.

Tuple spaces are a high level solution to coordinating processors in a distributed system whilst still remaining efficient relative to low-level solutions (Carriero and Gelernter 2006).

2.1.4.3.2 Channel Based

Channel based coordination models treat computation and communication separately by treating the output from one process as being the input for another. Processing elements communicate via message passing between one another over pre-determined channels. There is no shared memory whatsoever and processing units are treated as black boxes. The communicating sequential processes model (Hoare 1978) is the first significant channel based coordination model.

CSP is a very structured method of parallel programming. Programs are defined by parallel regions where processes are created and run until they terminate producing a set of outputs. These outputs can then be used as inputs for the next parallel region. In the simpler CSP case, processes identify each another by an explicit name and all read and write operations are synchronous and blocking.

The communicating sequential processes model also allows for the formal analysis of complex systems and prove them secure and free of defects such as deadlocks.

2.1.5 Problem Solving in Distributed Systems

In order to solve a problem concurrently, distributed and parallel systems decompose the overall job into smaller tasks that can be divided amongst numerous processing units or nodes in a processing network. The goal is to maximise the amount of processing whilst minimising the communication overhead. There are several

existing strategies in the literature to achieve this. Here will be described how a problem is commonly decomposed in a distributed processing system, how work is allocated and how subtask dependencies are managed. This is described with a focus on distributed architectures, and using that terminology, as there are many similar models such as data-parallel model (tasks are statically mapped onto processes, and the similar task is performed on different data), task-graph model (tasks are viewed via a task dependency graph), work-pool model (this involves dynamically mapping tasks onto processes for load balancing), master-slave model (one or more master processes generate work for other processes to do), producer-consumer model (a stream of data passes through numerous processes, and each one performs a task on it).

Sinnen describes the common methods by dividing them into the three overall categories of *data decomposition*, *task decomposition* and *hybrid decomposition* (Sinnen 2007).

2.1.5.1 Data Decomposition

Data decomposition is a common method of decomposing a problem for solving on a distributed computing system due to candidate problems often working with large datasets. The data are partitioned into subsets depending on the nature of the algorithm and distributed to nodes for processing. Input data, output data and intermediate datasets are all candidates for partitioning during the processing process.

Data decomposition is also used with other decomposition methods (the so-called hybrid decomposition) which is briefly explained in 2.1.5.3.

2.1.5.2 Task Decomposition

Task decomposition is used where concurrency can be found in the execution of tasks rather than the division of data. Common strategies include recursive, exploratory and speculative.

2.1.5.2.1 Recursive

This is the classical divide-and-conquer approach where a task is split into subtasks which form a tree structure. Each leaf of the tree represents an indivisible problem or a problem of suitable size for the processor's speed and communication overhead. Branches above leaves represent the sum of the dependent leaves below with the root being the solution to the problem.

2.1.5.2.2 Exploratory

Exploratory decomposition closely resembles data decomposition in the way that data are decomposed into a tree structure. However, it contrasts in the fact that not every node in the tree is computed as a solution. Decisions are made at each branch of the tree in order to come closer to the solution.

As a part of the exploratory process, it is possible to make a wrong decision and backtracking back over the problem can be an additional overhead.

2.1.5.2.3 Speculative

Speculative decomposition is suitable where a problem includes a set of steps where

at each step there are a number of possible actions to take. This is analogous to the switch statement in the C language. In distributed terms this means that tasks are organised and then a decision is made and a single task executed with the rest discarded.

2.1.5.3 Hybrid Decomposition

Hybrid decomposition is the combination of two or more decomposition methods in order to extract concurrency from a problem. A combination of task and data decomposition is a common solution to a variation of problems.

2.1.6 Task Allocation in Distributed Systems

There are multiple ways that tasks, once divided, can be distributed and allocated to processors in a distributed system.

Tree, crowd, economic and hybrid models will now be considered as examples of the major task allocation categories.

2.1.6.1 Tree

Tasks are allocated in a tree like structure by having a task start with a whole data set, then as another node joins, the set will be divided and a portion given to the new node. Should another node join, the task is divided again and so on.

This method of task allocation closely follows the channels of communication in the network of processors and closely relates to divide-and-conquer style of task decomposition.

2.1.6.2 Crowd

In contrast to tree computation, crowd allocation works by organising a pool of work and a separate pool of processing nodes and allowing the “crowd” of nodes to do the computation.

2.1.6.3 Economic

Some designs attempt to allocate a cost to each process in a computing network. See Rajkumar Buyya's PhD thesis for a comprehensive treatment of economy based task allocation (Buyya 2002).

Another variation on the economic theme is allocation based on reputation. Processing nodes may gain status and reputation through the amount of tasks satisfactorily completed. Here work may be allocated to nodes that are well regarded and reliable. Some systems may even lower the amount of double rechecking or re-computation that needs to occur to well respected nodes in order to manage confidence in the computed dataset versus the amount of time for completion.

2.1.6.4 Hybrid

A hybrid approach is simply a combination of two or more of the aforementioned schemes being used in concert for a desired result. Crowd computation along with reputation is commonly used as is a mixture of tree and crowd computation.

2.1.7 Task Dependencies

In concert with task/data decomposition and task allocation, task dependencies, for certain classes of algorithms, are of prime importance. Inter-task and parent-child task dependencies often need to be satisfied in order for an overall job to be completed. Mapping an arbitrary number of sub tasks and dependencies across an arbitrary number of processors in the most efficient way is an NP-hard[6] problem. (Garey & Johnson 1979).

There are several approaches to attempting to optimise this process. Here will be considered the common ones of *task graphs*, *Petri Nets* and *heuristics*.

2.1.7.1 Task Graph

Task graphs are typically directed acyclic graphs where each node represents a subtask that needs to be performed. Edges between nodes indicate order precedence and can have communication costs associated with them as well.

Various heuristic algorithms exist that can be applied to the graph in order to maximise efficiency.

2.1.7.2 Petri Nets

Dependencies can also be modelled using Petri Nets which are an extension of the task graph idea. Petri Nets have the advantage over directed acyclic graphs that they support loops and the ability to make choices at an execution point. They do this by describing processes beyond just the tasks and their hierarchy of task graphs. Petri Nets can be represented graphically, as a flow chart, or can be mathematically represented in a language.

See Bratosin et al (2007) for more information.

2.1.7.3 Heuristics

Several heuristic methods exist that, whilst quite effective in experimentation and practise, are not or have not been proven correct. Like all heuristics, these often provide good results when there is a large body of statistical analysis supporting the fitness of the heuristic to its particular application.

2.1.7.3.1 List Scheduling

List scheduling algorithms work on the idea of tasks being placed in different priority queues based on their importance. Importance may be measured by the number of other tasks that depend on this task being completed or the number of connected task that have already been scheduled and several other variations of this theme.

2.1.7.3.2 Graph Decomposition

The aim of graph decomposition is to identify sub graphs of tasks that correspond to an appropriate balance between processing time and communication overhead. It attempts to achieve an optimal grain-size for allocation of work to processing nodes (McCreary et al 1993).

6 “A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder.” (Weisstein 2009)

2.1.7.3.3 Critical Paths

Where costs have been associated with each processing node in the task graph, along with costs for communication along each connecting edge, a critical path is said to be the most expensive route from the root of the tree down to the bottom of the tree where the computation finishes. Critical path heuristics aim to shorten the critical path as much as possible by grouping adjacent nodes together; thereby removing the expensive communication links or edges in the graph. This produces groups of tasks which form a processing grain which can then be allocated to a processing node.

See Khan (Khan et al 1994) for an interesting comparison of five common multiprocessor scheduling heuristics, of the categories mentioned here, applied to a large set of program dependence graphs.

2.1.7.4 Genetic Algorithms

Genetic Algorithms are a specialised heuristic, based on the evolutionary processes of nature, that allows a search space to be examined in a polynomial time (Holland 1975). Searches are performed iteratively with information from previous searches being passed down the generations with random mutations in parameters along the way.

2.1.8 Using Distributed Computing for Large Problem Solving

Now that the major components and techniques of distributed computing are understood, the motivations behind projects using distributed computing over some more traditional alternatives like a supercomputer will be discussed.

2.1.8.1 Advantages

A distributed computing cluster, especially those comprising of cheap or existing machines, provides access to computing power that is much cheaper per MFLOPS compared to a similarly powered supercomputer. As the field has progressed, and through the Internet, access to many more participants, distributed computing has ceased being a lesser alternative and for many things has become the only feasible alternative.

The sharing of resources and replication inherent in many distributed computing systems can be a desirable trait all by itself. This is especially prevalent in another sub field of distributed computing called grid computing and the newer, and more nebulous, cloud computing.

Even machines permanently dedicated to the task can often provide comparable performance than one dedicated parallel machine at a reduced cost. The nodes in the network can change over time with machines added or subtracted as budget or availability of resources dictates.

Some problems are inherently parallel and/or inherently distributed. Some modelling, simulation and sensor systems require that their components be distributed. Robustness and availability is also in a similar vein.

Avoiding waste through the scavenging of computing cycles is fast becoming a consideration in recent times. These wasted cycles running an intensive screen saver[7] can amount to carbon in the atmosphere. If the machine is on and consuming resources it should be doing something useful!

7 It's ironic considering it's no longer the screens that need saving when a computer is left on.

2.1.8.2 Disadvantages

Distributed computing introduces many new problems that are not usually considered an issue in a single machine or dedicated computer cluster on a private network. Security is the main issue here. Can the computed data be trusted? Are all of the nodes well meaning and trustworthy?

As well as malfeasance, there is generally a higher level of failure of nodes than which is observed in a dedicated parallel machine. Nodes may come and go (this is known as *churn*). Nodes and their connections in the processing network will not be homogeneous.

Some algorithms simply do not scale well over comparatively slower networks. Even gigabit speed networks pale in speed comparison to processors existing on a purpose built bus to say nothing about keeping all of the processing on chip on a fast processor. This limitation is known and measurable, as already discussed, and is a big consideration when designing a distributed computing project.

Needing to know a lot about the nature of the computing job can be a significant problem. This again usually relates to the communication overhead where a fine grained problem with a lot of communication overhead can quickly render a distributed computing solution over a wide area network untenable. Indeed some problems will present too much overhead even if all of the participating machines are all on the same gigabit network.

2.2 Peer-to-Peer networks

Peer-to-Peer networks are characterised by each node in the network (peer) being equivalent to each other node in the network. This contrasts to the client/server approach where servers provide services to separate clients which consume them or control clients outright. Client/server approaches tend to have the server and client applications separately and independently written. In a peer-to-peer approach the client and server software is the same application. Each node in a peer-to-peer network acts as both a client and a server, consuming resources whilst also providing them.

Peer-to-peer networks are gaining popularity due to their robust, decentralised nature, low cost through usage of existing resources and potential for providing vast resources of computation and storage. Peer-to-peer networks have been in operation, in one form or another, since the earliest days of the Internet and have recently proven popular as a platform for the sharing of files on wide area networks. Peer-to-peer implementations also cover areas such as distributed communication, computation and storage.

2.2.1 Characteristics of Peer-to-Peer Networks

Peer-to-peer networks consist of a network of nodes that work together in order to provide services to users.

The *network* typically sits on top of the application level of the OSI model[8]. This is why many peer-to-peer networks, especially structured peer-to-peer designs, are known as overlay networks. Each network consists of a varying number of nodes.

8 The Open Systems Interconnection Basic Reference Model (or as commonly referred to the OSI Model) is an abstract representation of the various layers of communication and hardware that represent modern network architecture. See the OSI reference model (ITU 1994) for a complete reference.

Each *node* is typically an end user's computer connected over the Internet that is running the particular peer-to-peer application. In publicly accessible networks, nodes vary greatly between each other in terms of capabilities such as available computing power and underlying quality of their connection to the Internet (in terms of available bandwidth, latency and throughput). Each node provides resources to the network depending on the kind of peer-to-peer application it is. These *resources* usually include processing power, storage or disk space, provision of files, search or other services which together go towards providing an application to all users of the network. The nodes in the peer-to-peer network typically maintain the network by making a number of data connections with other nodes on forming the overlay.

2.2.1.1 Basic Peer-to-Peer Network Structures

There are a variety of network structures which govern how nodes and resources are arranged though generally they fall into the two broad categories of structured and unstructured networks. A *structured* peer-to-peer network has an order to the arrangement of nodes and/or resources in the network. Connections between nodes can be directed in order to improve performance at the cost of extra overhead in the network maintenance. In an *unstructured* network nodes and resources are not purposely arranged by any governing algorithm so nodes can make connections with other nodes arbitrarily. As with most things, there is the third option between either extreme which can be defined here as a *partially structured network*. This is commonly expressed in peer-to-peer networks as a heuristic approach to limit some of the excesses of a completely unstructured approach. Common examples of this will be self arrangement of nodes based on their network connectivity (i.e. Conspiring to move slower nodes to the edges of the network) or active or passive attempts to remove obvious flaws in the overlay network arrangement such as cycles in the graph etc.

2.2.1.2 Centralisation

In addition to structure, peer-to-peer networks can be centralised, semi-decentralised or decentralised. A *centralised* peer-to-peer network will have one or more network services hosted on a centralised server separate from the rest of the nodes. The *decentralised* network has no central elements. All network services are distributed across the nodes in the network. The notions of centralised and decentralised networks is an important classification for peer-to-peer networks. The *centralisation* of a peer-to-peer network refers to how many non-peer elements exist in the design. Totally decentralised networks require every single peer to have an equal role in the network without any central elements at all. A purely decentralised network is will have every node equivalent to another with no central or independent mechanisms. Completely decentralised peer-to-peer networks are rare with most having varying degrees of centralisation and imposed structure and/or roles.

2.2.1.3 Bootstrapping

Bootstrapping is where a system initially starts a simple process whose purpose is to start a much larger process. In peer-to-peer terms, bootstrapping often occurs where a node needs to join the overlay network. It usually consults a database of known nodes' IP and port details from previous connections or from known centralised repositories of node information. This is a classic causality dilemma (the proverbial chicken or egg problem), which in peer-to-peer terms always introduces at least a small part of centralisation in order to maintain this information. Practically however,

as networks grow, the centralisation becomes less of an issue as you have private databases of known nodes on the network and a multitude of online databases which are unlikely to all be compromised at the same time.

See Karbhari et al (2003) for a study of bootstrapping in peer-to-peer systems that includes a more detailed description.

2.2.2 Peer-to-Peer Network Architectures

Many early examples of peer-to-peer networks and protocols were never thought of in quite the same way that modern peer-to-peer systems are thought of. Even though connecting machines together not as clients and servers but as equal peers dates back to the original ARPANET, it wasn't until 1998 that an application (Napster) was written that is P2P in the modern sense (Magoules et al 2008). Between these two times, the mid 1960s and 1998, many applications were written that are quite peer-to-peer in their nature. It may seem, with hindsight, that a P2P solution was a natural fit for some problems and solutions were written without much thought given about this being a new technique in itself. This is interesting when one considers that some of the most decentralised peer-to-peer examples are actually within these early systems. Such examples are covered further in the next chapter.

These early systems are actually represented in some classic Internet protocols as a kind of “store and forward”. These peer-to-peer systems were extremely popular and effective as they essentially batched communication between peers at certain times. The obvious example of this is the Usenet protocol NNTP. Each host acted as both client and server moving data over large capacity links at certain times of the day. This allowed for relatively efficient communication between peers and limiting the “uncontrolled” bandwidth usage to the edges of the network where a news client would connect. The edges of the network were client/server but the primary network is peer-to-peer.

This section focusses on the underlying techniques used by peer-to-peer systems to give an understanding of the current state of the art in both commercial and research systems. It covers the main four approaches of *flooding systems, partially centralised, hybrid systems* and highly structured systems based on *distributed hash tables*.

2.2.2.1 Flooding Systems

Flooding systems work by having nodes build their own network though means of needing to know of a connection point or existing node on the network in order to join. These connected nodes then form an overlay network over the Internet. A list of known nodes (also known as hosts) would be forwarded between nodes on the network as well as the payload data that the network was tasked with sharing. This required that a host joining the network would need to know of another existing host on the network in order to connect and obtain, and then add itself to, the host list (the so called *bootstrap* process).

Smaller peer-to-peer systems have also relied on network discovery based on broadcast to bootstrap themselves onto the peer-to-peer network. A discovery broadcast message could be sent to all known machines on another service.

Once a suitable peer-to-peer connection candidate has been established, the new node attempts to communicate with it. A handshake process is then undertaken where the nodes exchange messages to establish a peer-to-peer connection. These

messages typically consist of a connection request ("client" side), connection accept or connection refuse ("server" side). The nodes will often exchange information regarding other nodes that they are connected to, nodes that have been connected to previously, and other suitable information to allow the overall network to grow.

Once a connection has been established, in addition to application specific information being exchanged (such as search packets in a file sharing network), nodes will exchange keep-alive messages periodically to detect connection failure. These messages are sent at an interval as a trade off between the time difference between discovery of failed nodes and the bandwidth required for these messages. These systems often suffer scaling problems from the amount of bandwidth required and various strategies are undertaken to try to mitigate this obvious limitation at the expense of network purity.

2.2.2.2 Partially Centralised Systems

As discussed previously, flooding of packets as the main purpose of an overlay network to fulfil its requirements, is inefficient and limited in scalability. An obvious solution, is to partially centralise the overall system in order to remove the flooding aspect. Peer-to-peer file exchange networks that have two main functions, search and file exchange, have moved the search and node discovery functionality to a centralised index whilst leaving the file exchange to be between peers.

Partial centralisation will solve many of the problems associated with completely centralised peer-to-peer systems at the expense of the associated costs it brings such as a necessary investment in infrastructure and a single point of failure.

2.2.2.3 Hybrid Systems

Hybrid systems attempt to have the best of both worlds when it comes to pure decentralised peer-to-peer and partially centralised peer-to-peer. Most systems try to work well without a centralised organisation element, but work better with one. Also, they are designed so the centralised elements are non-specific and more of a protocol rather than a single installation. That way, swarming hybrid systems can rely on many different single centralised servers or on several different ones at once.

2.2.2.4 Distributed Hash Tables

Of the very structured and decentralised peer-to-peer networks, the distributed hash table approach is still where most research efforts are concentrated.

A *hash table* is a data structure that associates keys with data. This means that a piece of data can be placed into memory and can be later recalled using the key that it was associated with. Data is recalled later through a lookup function which usually takes the key as a single parameter and returns the earlier stored value. A variety of algorithms exist for arranging the data within memory so that keys can efficiently and uniquely recall data. A *distributed hash table (DHT)* extends the process a step further to allow the hash table to be distributed over a network of nodes yet still allowing for the simple lookup interface of a traditional hash table. This is achieved by a technique known as *keyspace partitioning*. This means that each node on the distributed network becomes responsible for a range of keys within the keyspace of the hashing algorithm. So that when a lookup is called it must first be determined which node is responsible for the key before doing the last lookup step of the traditional hash table. This means that nodes in the network will be arranged in such

a way that the key can address both the node as well as the item in the hash table. This is usually achieved through the use of a routing table so that a search for an item by key will result in a traversal of the network, each step coming closer to the node that actually holds the correct key. This results in a reasonable certainty of failure and search time that was so lacking from the flooding techniques in second generation P2P networks.

DHTs can be implemented using a variety of algorithms. The examples given here are the three most popular variants based on trees, skiplists and the Cartesian coordinate space.

2.2.2.4.1 Trees

Distributed hash table based peer-to-peer systems have been based on several different tree variants for their general organisation structure. Binary trees, B-Trees and Plaxton trees have all been used to organise search spaces and/or communication.

Binary trees, and their common variants (See Sedgewick (1990) for a resource on tree implementations), are used to organise a hierarchy of nodes and data keys to maintain an overall logical network A Plaxton tree (or mesh), is an interesting tree variant where a property of the tree structure is that every node is also a root of its own tree (Plaxton et al 1997).

2.2.2.4.2 Skiplist

A Skiplist is a probabilistic structure that essentially consists of several layers of ordered lists. It has roughly the same insertion times as a B-Tree implementation yet is simpler to implement and maintain for certain tasks (Pugh 1990). There has been much interest in peer-to-peer research with several projects basing their implementations on skip lists or skip graphs.

2.2.2.4.3 Cartesian Coordinate Space

Another variant for maintaining a DHT is the Cartesian coordinate space approach. Here the hash table is spread over a geometric area with each node being responsible for an area of the graph.

2.2.2.4.4 DHT Summary

DHTs are an active research area with several new systems emerging each year along with further study and refinement of earlier proposed systems based on the underlying algorithms mentioned here. There are other systems based on rings, butterfly networks and de Bruijn graphs, however the three general approaches of trees, skiplists and Cartesian Coordinate Spaces provide enough background for this thesis. See Risson and Moors (2004) for research leads on these extra systems. Chapter three of this thesis will examine a number of implemented examples of DHT based peer-to-peer systems.

2.2.3 Advantages of a Peer-to-Peer Approach

Avoiding financial cost is a major factor in favour of peer-to-peer networks. Immense storage is available and evenly distributed amongst participating nodes rather than being concentrated to a few powerful servers. Also the costs of bandwidth are spread over the network. P2P distribution networks deliver vast amounts of

bandwidth that would, if implemented using FTP on a single site, cost a significant amount for necessary hardware and network service. The motivation of this thesis, and of several other recent contributions, has been to leverage this power for computation. Decentralised P2P computing has already proven itself to provide an immense amount of ongoing processing cycles for a fraction of the cost of dedicated systems.

Capacity automatically increases with popularity. Well designed P2P systems allow the overhead of extra users to be mitigated with the extra resources they provide. This contrasts to a centralised system where there is a finite server capacity and that every extra user consumes part of this resource without giving any back.

Centralised systems also present a single point of failure and weakness to attack. A distributed system has the potential to be a robust system as load and risk can be spread relatively evenly over the entire network.

2.2.4 Disadvantages of a Peer-to-Peer Approach

Distributed systems have their own set of vulnerabilities too especially in regards to scalability and security. Many P2P systems suffer from design problems. As already discussed, central elements critical to a P2P design have shown that a system with millions of users can be brought to a halt by an external issue that is not a design or operational fault (ie. legal action). It has also been discussed that some systems that have relied on flooding queries have been shown to be ultimately infeasible as the network load increases.

Security is a problem of general concern for any network; in the case of P2P it is especially so. Depending on the application, issues of non-repudiation and trust are especially important and difficult to implement correctly in a dynamic environment. Conversely, in other applications anonymity is the greatest concern, and like non-repudiation, difficult to implement efficiently and correctly. Attacks also form a problem. In most P2P networks denial of service (DOS) and flooding attacks are common. Bad nodes can flood another node with many superfluous messages or connections in order to consume the target node's resources and weaken the network at that point. In file sharing networks, poisoning is also a problem. Files containing misrepresented data are often left on the network. Another user may download one of these decoys only to find the file contents are not what they had expected. This consumes network resources and affects a user's trust in the system. These are just some of the security issues that need to be considered when designing and implementing a P2P system.

2.3 Benchmarking and Evaluation

Benchmarks are standardised programs, methods or specifications designed to assess the performance of a system. In computing, there are several different types of benchmarks such as a *micro-benchmark*, *kernel*, *synthetic* and *application*. Each of these can be applied to a system to gain insight into its operation compared to other similarly tested systems. A micro-benchmark is used to assess a basic component of the system such as speed of a particular class of instructions or a mix of different instructions. A kernel benchmark is based on what previous researches have considered to be useful or common workload for computing systems. Sorting, matrix operations and searching algorithms require a mix of instructions that can better assess a system's usability over just the speed of some individual instructions alone. A synthetic benchmark is typically a program made up of instructions derived from a

statistical analysis of what the computing system is doing during real work in a variety of applications. And, finally, an application benchmark is one that is based on commonly used applications that provide a useful comparative metric. Operations times in common business applications have proven popular as have more modern examples such as frame rates in popular video games, time to compile widely available code bases or render a ray tracing scene. In each case an application benchmark takes advantage of both an application's nature of computing usage and it's ubiquity.

Here will be examined the major areas of evaluation of computer systems, in particular, those techniques that apply to distributed and parallel systems. Section 3.3 will discuss specific projects, limiting the discussion to the underlying techniques starting with the foundations of machine performance, then measurement techniques for distributed systems and finally an examination of methods applicable specifically to peer-to-peer systems.

2.3.1 Machine Performance

Machine performance consists of several different areas and techniques. This section begins by examining the historically important ones based on instructions and floating point operations per second, benchmarks and simulation.

2.3.1.1 IPS and FLOPS

Early computer measurements were often based on the number of instructions per second (IPS) that a machine was capable of. Whilst useful for selling computers (“this machine does 500 more instructions per second than our closest competitor”), it doesn't provide much of a metric for evaluation. Similarly a more modern example would be to compare processors of different manufacture by clock speed alone.

Instruction mixes soon appeared which are statistical approximations of the kinds of instructions that a machine would do over a period of time. A mix could then be derived and ran on a machine in order to gain an insight into how it could work with a real workload. Mixes were required as it was common for instructions to have vastly different cycle times, or indeed number of cycles required, between machines of different manufacture. Without a standard mix, it would be easy to contrive a test to unfairly exploit a small advantage one machine may have whilst covering its many disadvantages.

For many scientific applications, floating point operations tend to be almost exclusively used, so a measure of floating point operations per second (FLOPS) was considered enough of a measure for these generally expensive operations.

2.3.1.2 Benchmarks

Benchmarks are a set of standardised trials, usually in the form of software packages, that are executed on a computer to assess its overall performance. They can also be expanded to apply to software or hardware testing in order to test for correctness of implementation. Ideally a benchmark should be open, standardised and independently verifiable. Still, care needs to be taken in the selection of a benchmark with an understanding of the hardware being measured. Modern pipelined systems may not perform well on synthetic benchmarks.

Benchmarks themselves fall into a number of separate categories that will now be examined in turn.

2.3.1.2.1 Synthetic

These benchmarks closely correspond to the mixes discussed previously. A statistical analysis of a machine's operation is taken over a period of time to gather data on the percentage of each instructions usage whilst under load. An application is then written to perform these instructions in the same proportions.

2.3.1.2.2 Kernel

Rather than just the individual operations of the synthetic benchmark, a kernel benchmark looks to the algorithms that are commonly used in order to gauge a machine's performance. Mathematical subroutines such as linear algebra are often used as a basis for performance measurement.

2.3.1.2.3 Component and I/O

Component benchmarks individually measure the performance of each part of a computer system. Synthetic and Kernel benchmarks will often concentrate only on processor speed. A component benchmark will also assess the speed of associated hardware such as memory, cache, register speed and so on. Input / Output (I/O) benchmarks will also assess networking and storage devices.

2.3.1.2.4 Real Software

Software applications are often used as de facto benchmarks for modern computer systems. The number of frame rates that a video game can display per second, can be a good approximation of graphics and processor performance. Compiling a large, widely available code base can also give a good indication of processor, memory and disk performance. Commonly available business applications often serve as a good benchmark due to their load and especially their ubiquity allowing for ad hoc measurements to be applied without installation of specialised software.

2.3.1.3 Simulation

Simulation is used to assess performance where the system being measured is still in design or where it is not practically feasible to reproduce that actual environment in order to perform an actual test. Many modern systems based on distributed processors or that operate over wide area networks or load of many thousands or even millions of discrete users, make actual testing difficult if not impossible. Applications are therefore written to produce a simulated load or simulate the system itself that is to be tested.

The February 2002 issue of IEEE Computer (IEEE 2002) is a special issue dedicated to performance simulation and provides many quality leads to further information.

2.3.2 Distributed and Parallel Systems

Benchmarking and evaluation of distributed and parallel systems use combinations of all of the techniques presented in machine performance in the previous section. However, there are a few variations specific to distributed and parallel systems; especially wide area network distributed systems.

2.3.2.1 Speedup and Efficiency

As already discussed in 2.1.2, speedup and efficiency are two common goals of distributed and parallel systems. It is not unusual to expect then that these form the base measure of many benchmarks for distributed systems. The actual benchmarks themselves will be a suite of kernel or synthetic benchmark applications however the addition of speedup and efficiency as the application scales over the processors is often of equal interest. This is especially the case to designers of new overlay networks where the speedup is the primary way of evaluating a new design.

2.3.2.2 Statistics and Leaderboards

A wide area network peculiarity is the so called leaderboards that occur for contributors to large distributed computing projects. Performance of user's machines (or clusters of machines) are displayed on public leaderboards. These often show the number of work units completed and are an indicator of individual node performance as a part of the computing cloud or grid.

2.3.3 Peer-to-Peer

Peer-to-peer benchmarking is still in its infancy. There has been some work done in this area yet it still has a long way to go before any standards emerge. What benchmarks do exist are primarily based around what would be considered, in other performance measurement areas, application benchmarks. When considered carefully, this should not really be surprising considering that the vast majority of peer-to-peer implementations are generally at the application layer of the OSI model. So, that being said, the following section will consider several of the areas of peer-to-peer that have had some performance measurement applied across implementations before looking at some specific examples in chapter three. Chapter Four endeavours to make some new additions to benchmarking of peer to peer systems by considering several issues that have not been addressed by the available literature.

2.3.3.1 Search Performance

Many peer-to-peer applications rely on finding keys. DHT based designs are keyspaces partitioned over nodes. Finding who owns a particular key differs greatly depending on the underlying algorithm chosen and its implementation.

2.3.3.2 Replication

Decentralised systems rely on replication in order to deal with node churn. Different algorithms will have a marked difference on the amount of time and bandwidth consumed to find and maintain replicas of data.

2.3.3.3 Overlay Performance

Managing and maintaining the overlay will again differ by the choice of underlying algorithm and the implementation between systems. The amount of bandwidth consumed by keeping the network together and performing housekeeping tasks has been the subject of analysis in the literature. It hasn't gotten as far as being incorporated into any kind of comprehensive suite, however some individual metrics have been identified. Examples of these metrics are discussed in section 3.4.6.

2.4 Chapter Summary

This chapter has reviewed the separate areas of distributed computing, peer-to-peer networking and measuring the performance of computing systems. It concentrated on the theoretical background and underlying concepts with examples only where necessary for explanation. Chapter Three continues by showing where examples of these technologies are used.

Chapter Three

3.0 Related Work

This chapter builds on the overview of technologies discussed in the previous chapter by presenting a survey of some representative systems that best provide context for the contributions of the remainder of this thesis. The chapter is divided into three sections covering peer-to-peer networks, distributed computing systems and previous work in the benchmarking of distributed computing systems and peer-to-peer performance.

3.1 Peer-to-peer networks

Many early examples of peer-to-peer networks and protocols were never thought of in quite the same way that modern peer-to-peer systems are thought of. The term peer-to-peer did not emerge until well into the 1990's. It seems, with hindsight, that a peer-to-peer solution was a natural fit for some problems and solutions were written without much thought given about this being a new technique in its own right. This is particularly odd considering that many early examples were comparatively sophisticated (c.f. some modern, successful peer-to-peer systems) and "more peer-to-peer" if you like than many first and second generation P2P examples given later.

The following section will consider some early protocols and systems that exhibit strong peer-to-peer traits spanning from the late 1970's up until the late 1990's. From there it follows on to examine the first widely recognised peer-to-peer systems that emerged in 1999 through to today.

3.1.1 Classification of Peer-to-peer Systems

Given that peer-to-peer is still a relatively young area of academic interest, with the first major conferences appearing around 2001 (O'Reilly 2001), there is still no clear set of categories with which to describe the classifications of peer to peer networks. Some describe them via application and capabilities (i.e. Milojicic et al 2002) and many historically around software releases e.g. 1st generation, 2nd generation and so on (i.e. Eberspächer & Schollmeier 2005). Furthering this evolution here presented is a 4th generation classification, expanding on the earlier evolving classifications.

New Classification	Historical/Generative Classifications	Classification by Example
Store and Forward	Early, Historical	SMTP, NNTP
Centralised	1st Generation	Napster
Decentralised	2nd Generation	Gnutella, SMB
Structured	3rd Generation	Pastry, Chord, CAN
Unstructured	2nd Generation	Gnutella
Hybrid	Hybrid	Bittorrent

Table 2: A new classification scheme by peer-to-peer approach along with its generative classification equivalent and examples.

This simple taxonomy should aid the reader to keep track of the multitude of descriptions given in the referenced literature. It is also the furtive hope of the author that this new and modest classification scheme may find some traction with authors of new papers in a bid to find some standardisation as even new papers continue to use different mixes of classifications in their literature reviews; especially the outdated generative model which becomes continually less relevant as newer implementations pick from a variety of functionalities across these 3 or 4 generations.

3.1.2 Store and Forward

"Store and Forward" peer-to-peer systems are those that act as both client and server for communication between peers over the Internet. If discussion is limited to applications that exist on the Internet, and not spiral recursively back to the Internet itself and to UUCP[9] before that, two popular early core applications of the Internet were very peer-to-peer in their nature without actually being remarked as such explicitly (certainly at the time anyway). One is NNTP and the other is SMTP and both are discussed here.

3.1.2.1 NNTP

The NNTP is the protocol used by the USENET which is one of the first widely used distributed systems that has been lately generally accepted as an early example of P2P. The USENET, conceived in 1979, is a public message system that allows users to post and read messages under a variety of different categories. Each USENET server is a peer on the USENET network and messages posted to a peer are shared or replicated around to all of the other accepting peers on the network. In modern USENET each peer speaks a protocol called NNTP which allows a message to be uploaded in one place and then quickly propagated over the entire USENET network. It is to be noted that while the system and network itself is very P2P, the access of it

9 **Unix to Unix Copy Program (**UUCP) was an early application that allowed for commands to be run remotely and for files to be transferred between computers. It is a classic example of a store and forward architecture. See RFC976 (Horton 1986).

is not. At the fringes of the network users interact with the USENET server using a separate client application in a very client/server (read: non P2P) manner. This can also be compared with electronic mail.

3.1.2.2 SMTP

Email, at its core, is a group of servers quite similar to USENET which forward email messages between servers with client applications being involved only at the fringes. The Simple Mail Transfer Protocol (SMTP) is the enabling technology. Each email server acts as both a server and a client to all of the other email servers on the Internet, accepting email messages and forwarding them along to the next closest server to the addressee.

See RFC821 (Postel 1982) and RFC5321 for the original SMTP and Extended SMTP (Klensin 2008) respectively for an exhaustive definition of these protocols.

3.1.3 Centralised

Some peer-to-peer systems are only partly peer-to-peer, that is they contain a significant centralised component without which they could not operate at all. The most classic example of this is Napster.

3.1.3.1 Napster

Napster was an infamous file sharing application that was first released in mid 1999. It allowed users to host their files, search for files hosted by other users and subsequently trade files between each other. The Napster network was predominately used for sharing MP3[10] files.

Napster's design was not very decentralised from a P2P perspective. The file transfer is between peers, however the searches are submitted to central servers. This provided a simple solution to the problems of distributed search (by avoiding it altogether) but created a single point of failure. When this central index was shut down, as it was in July 2001, it rendered the system inoperable.

3.1.4 Decentralised

Decentralised peer-to-peer systems are those that try to avoid all centralised elements as much as possible. Gnutella is the most obvious example of a decentralised peer-to-peer network.

3.1.4.1 Gnutella

The second generation of P2P file sharing networks evolved as a response to the weakness of the centralised elements of Napster. The natural focus was to remove the central search servers that made Napster vulnerable. Gnutella was the first such example.

Gnutella is a protocol for distributed file sharing and search. The file exchanges work much the same as Napster, with peers exchanging files between

10 MPEG-1 Audio Layer 3 is a common audio file format that employs lossy compression techniques to reduce the size of recorded audio at the cost of removing very high and very low frequencies that typically cannot be heard by humans.

each other, however the search functionality is decentralised. Early versions of Gnutella worked by having searches propagate across the network. Nodes submit searches (usually just a search term string) and then pass this search packet on to all of the other nodes that they are connected to. This flooding approach allows nodes to see all of the search packets that come their way and notify the originating node (search packets also contain address information) if a match occurs. The approach, whilst very hard to "kill", has several other obvious design limitations. The flooding approach to search does not scale very well and the open nature of search term propagation is open to malfeasance. Enormous amounts of bandwidth can be generated across the network as nodes forward one packet to every connected node and then they forward it to every node they are connected to and so on (Ritter 2001).

3.1.4.2 Server Message Block

Another ubiquitous protocol is the Server Message Block (SMB) protocol which is more commonly regarded as the network protocol used by Microsoft Windows for peer networking or "Workgroups" between machines. Each machine that is registered on the network acts as a peer to all others. File can be requested or served from each machine. Its management is decentralised as each machine can largely choose what is to be shared and what is not. This contrasts with many other file protocols, which are strictly client server such as the File Transfer Protocol (FTP).

SMB was originally developed at IBM and widely implemented by Microsoft and is very much in common use. The Microsoft implementation has been subsequently reverse engineered by the SAMBA team making it widely available and interoperable with other platforms (Ts et al 2003).

3.1.5 Structured

After the popularity of Napster and Gnutella, by the late 90s and early 2000s academia recognised the potential for these new designs as fertile ground for research. The distributed search problem was vigorously investigated and several variations based on classic data structure of a hash table were presented in papers. These extensions to the hash table idea (as discussed in Chapter Two) culminated in several different designs in 2001 which together represents the third generation of P2P design.

3.1.5.1 Distributed Hash Tables

The designs and implementations of DHTs that appeared somewhat concurrently were Chord, Pastry and the Content Addressable Network. Each system maintains its own overlay network and provides a hash table interface to storing and retrieving data. Each is based around a different data structure idea for organising its nodes in the overlay.

3.1.5.1.1 Chord

Chord (Stoica et al 2001) is one of the earliest DHT implementations. Chord is based around a skip list (See 2.2.2.4.2) where nodes are arranged in a circle with pointers forwards and backwards to other nodes as a successor and predecessor in which Chord calls a *finger table*. A hash table is distributed over this arrangement of nodes.

Chord provides a single lookup operation that when given a key it will find the node on which the key resides. Chord manages the underlying work of ensuring

that key integrity is maintained as nodes churn through the system by joining and leaving through maintaining lists of successors and predecessors as well as replication of keys throughout the system.

The Chord system is designed to be an underlying architecture for applications to be built on. There have been several prototype systems developed to demonstrate different potential usages. These include a cooperative file system, a distributed USENET cache and a distributed version of CiteSeer[11] called OverCite (Chord 2008).

3.1.5.1.2 Pastry

Pastry (Rowstron & Druschel 2001) is similar to chord in terms of overall design, including arranging nodes into a ring with links forwards and backwards between nodes. It contrasts when it comes to implementation. Pastry has its overlay network built on top of, rather than a part of, the distributed hash table implementation. This modularity allows for its routing metrics to be supplied by an external application. This can be used to optimise communication between physically close neighbours.

Implementations of systems using Pastry include the PAST distributed file system and the SCRIBE publishing and subscription system.

3.1.5.1.3 Content Addressable Network (CAN)

Of the three systems described in this section, CAN (Ratnasamy et al 2001) is the one which contrasts most in underlying design. In CAN data are stored in a multi-dimensional Cartesian co-ordinate space which is divided into different regions (called zones) with each node in the overlay network responsible for a given region in this space. When a new node joins the system it picks a random point in space and initiates a request to join the zone that this point resides in. When a successful request is made, the joining node will receive half of the zone region that its destination node originally managed. Joining nodes keep making requests until they have been allocated a zone.

Each CAN node maintains a list of the zones that neighbouring nodes manage. When a routing request is received, the node will determine which zone is closest to the zone containing the destination point and then forwards it to that node's IP address.

CAN manages node churn through periodically sending heart beat messages. When a node is deemed to have failed, the zone it was responsible for needs to be absorbed by the rest of the system. If the zone can be added to an existing zone they will be merged, if not, the zone, in its current geometrical form, needs to taken over by a node. Nodes periodically examine the fitness of zones in their control for merging. When zones are changed, messages are broadcast to neighbouring nodes so their zone tables can be updated.

3.1.6 Unstructured

Both Gnutella (3.1.4.1) and SMB (3.1.4.2) fit into this category as well as they are both decentralised in their operation and administration whilst also being unstructured in their organisation. Beyond file sharing there are also examples of distributed operating systems, considered as such since the provided services beyond just file sharing. There were several examples of workstation systems in the early

11 CiteSeer is a library of computer science publications and citations. See http://citeseer.ist.psu.edu/

1980s that were based around P2P networks. One such was the Convergent Technologies Operating System. This OS ran on early workstations based around Intel processors that were networked together into small clusters with RS-422 serial cables (Centre for Computing History 2009).

3.1.7 Hybrid

The final category is the catch all which relates to combinations of centralised and decentralised, structured and unstructured as well as some of the earlier elements of store and forward. The costs of purely centralised elements (single point of failure) versus the cost of an entirely decentralised approach (extremely high overhead with potential poor quality of service) has led many designers to look for a compromise approach.

3.1.7.1 BitTorrent

BitTorrent is a novel approach to file sharing that aims to maximise the distribution of data in terms of bandwidth and speed. When downloading a file with traditional FTP, a user only uses the download channel of their Internet connection whilst the upload channel is mostly unused. BitTorrent takes advantage of this and allows other users to download parts of the same file that another user is downloading at the same time. While a user is downloading a file they may also be uploading a file to another user using this otherwise unused upload channel. This balances the load of the file distribution bandwidth across all of the peers and reduces the load on the originating server. The original node need only upload one copy of the file being shared rather than FTP where the server needs to upload every copy requested by a client.

To use BitTorrent for downloading a user must first find a file which gives the necessary information about the swarm the user intends to join. As BitTorrent does not provide its own search facility it relies on existing search methods available to Internet users to find a "metainfo" file which contains the information required to join a BitTorrent session. This metainfo file, or "torrent", is typically hosted on an ordinary web server and contains information about the file that is being downloaded, such as its length, name, number of pieces the file has been split into for distribution, hashes of these pieces, and the URL of a "tracker" to coordinate the swarm.

A BitTorrent tracker is a small HTTP based service that allows BitTorrent users using the same torrent to find each other. When each BitTorrent download is started, contact information about the new downloading node such as their IP address, BitTorrent listening port are sent to the tracker. The tracker then responds with the same kind of information about others who are downloading the file so the new node can attempt to make other connections and increase its download and upload bandwidth. New implementations of the BitTorrent protocol are also beginning to use distributed hash tables (DHT) in order to attempt a decentralisation of the tracker component. This centralised component, contrasts starkly to the centralisation of a system like Napster (3.1.3.1) as you can choose from a myriad of trackers, or start your own. This allows for easy fail over should one centralised service become compromised or unavailable.

BitTorrent also employs many techniques to ensure that the transfer between two nodes is as efficient as possible and that data is replicated as reliably as possible. Such techniques include pipelining of data requests, order selection of file pieces

and interaction management between peers.

3.2 Distributed Computing

There are a myriad of distributed, grid and parallel computing architectures that have been proposed, implemented and used over the last some 30 years. To survey them all would easily provide enough material for a textbook. Here we will primarily outline BOINC and Condor as the two distributed computing systems that represent systems that have strong academic roots and current widespread use in a variety of places. A brief survey is then conducted of distributed computing systems that are peer-to-peer in their nature and have shown operational evidence or academic review.

3.2.1 Client / Server Systems

3.2.1.1 BOINC

The Berkeley Open Infrastructure for Network Computing (BOINC) is a volunteer distributed computing system (Anderson 2004) that grew out of the earlier seti@home project (Anderson et al 2002). From its origins processing data from the Arecibo radio telescope, it has been generalised for use for a variety of projects. It is a client/server system that has been designed to take advantage of spare computing cycles on participants' computers.

A BOINC project is instigated by having the BOINC software installed and running on a server. The server is also furnished with a data set to be computed as well as the algorithm (or set of algorithms) for the clients to apply to the data set. BOINC can be used with existing applications that can be distributed as the computing algorithm for the clients to process work units with. Work units are distributed to clients as they request them and clients report back with results. Malfeasance is managed using public-key encryption and multiple calculation of work units by unrelated clients. BOINC has a default scheduling algorithm which matches the resources that a user has provided with the size of jobs available for computation in a first come, first served fashion.

BOINC is wide area network client-server distributed computing system with a focus on volunteer participation. It has a sophisticated client application which allows a user to chose to contribute to a variety of projects. The host of a project has many options for the customisation of aspects of the user interface to encourage participation. This usually includes a screensaver as well so that an appealing display is shown when processing work units. Also the “skin” of the application itself is programatically customisable to encourage adoption. In addition to visual appeal, there is also a credit system to reward the contributor with “points” or credits for verified results. This attempts to encourage participation through a leaderboard of high credit scores to advertise top contributors. Whilst, at time of writing, no work has been done to quantify these incentives, BOINC remains one of the most popular platforms for volunteer wide area network distributed computing.

Appendix C contains detailed instructions for setting up a computing project using BOINC. These instructions form part of the method used later in this thesis.

3.2.1.2 Condor

Condor is a software framework for distributed computing that was developed in the

1980s at the University of Wisconsin-Madison (Thain et. al. 2005). Condor can run both parallel and sequential coarse-grained jobs and is suitable for both dedicated clusters and spare processor cycles on desktop computers. Condor provides support for queueing jobs, applying scheduling and priority policies and the monitoring of computing resources and job completion.

Condor arranges different classes of computer groups into processing pools. Each pool advertises its availability to the Condor *collector* which serves as a database for available resources. Jobs are submitted by users and placed in a queue. A *matchmaker* takes jobs from the queue and allocates resources to them based on policies and the job's associated requirements. Job requirements are advertised using a job description language and processors are described also. The matchmaker pairs compatible jobs and resources and allocates the work.

Condor supports jobs that are not independently parallel. Dependencies between jobs can be specified by stipulating a directed acyclic graph which will ensure that jobs are ran in order. In addition, Condor supports both MPI and PVM allowing for communication between processes as they are being processed. Condor is a mature platform that integrates readily with other systems (such as Grid middleware) to provide integration into larger systems such as Grids.

Appendix C contains detailed instructions for setting up a computing project using Condor. These instructions form part of the method used later in this thesis.

3.2.2 Grid Computing

Grid computing aims to be analogous to the power grid system where significant computing power is cheap, easy to access and ubiquitous. It compares very closely to distributed computing and, in many implementations, is broader in scope as it attempts to provide all of the infrastructure required for many different kinds of general purpose supercomputing applications. To further differentiate it from classical distributed computing, a grid should not be administered centrally, it should be based on open standards and quality of service should be non-trivial (Foster 2002).

Here the Globus Toolkit, P-Grid, X-Grid and by way of helping clarity, Cloud Computing will briefly be discussed.

3.2.2.1 Globus Toolkit

The Globus Toolkit is a suite of open standards which provides middleware for a grid computing implementation. There are standards for architecture, job submission, web service, security and so on which can be used to interface with distributed and parallel computing systems. As an example, Condor (3.2.1.2) can be integrated into a Globus Toolkit system where Condor provides the processing power while Globus manages the job submission. See Foster (2005) for a more detailed overview of Globus Toolkit.

3.2.2.2 Other Systems: P-Grid & X-Grid

There are many other grid computing initiatives beyond just the Globus Toolkit. However, in many cases, it is difficult to differentiate them from distributed computing. In fact, applying Foster's check list yields fairly low scores.

P-Grid, which is discussed further in 3.3.1.3, is a peer-to-peer based grid system that has been the subject of academic enquiry. X-Grid (Apple 2007) is a proprietary

system for harnessing computers together from Apple Inc. It is based on the BEEP protocol (Rose 2001) which provides a full-duplex frame based communication method that can run over TCP.

Both of these systems purport to be grids in their own right, hence their inclusion here, yet in both cases they fail the three point test (Foster 2002) (In the case of P-Grid by not following open standards for grids and X-Grid by being centrally managed).

3.2.2.3 Cloud Computing is not Grid Computing

Cloud computing has gained much attention in recent times. It can be described as software as a service, where applications and services are provided via the Internet that would traditionally be locally hosted on a private machine. It is an emerging area that has many definitions where many are still being sought (SYS-CON 2008). Buyya et al's (2008) later definition is "A Cloud is a type of parallel and distributed system consisting of a collection of inter- connected and virtualised computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers." Many cloud computing applications would actually be provided on distributed hardware at the server level, however as a paradigm itself it is not distributed processing in the conventional sense. It is often centrally managed, centrally hosted and accessed in a client/server fashion albeit with web browsers typically (so it is not strictly client server in the traditional sense either). It is mentioned here as is is commonly confused with Grid Computing and it is an area where peer-to-peer techniques are likely to play a large part. However, confusion well aside, it is an application of distributed computing and worthy of mention.

3.3 Related Work Combining Peer-to-peer and Distributed Computing

Now that peer-to-peer and client/server style distributed computing applications have been described, this section will move to describe work that has already been done as a combination of the two fields. That is, similar work which directly relates to the application later presented in this thesis.

3.3.1 Retrofitting BitTorrent into Distributed Computing Data Distribution

There has been some considerable work done in replacing the file distribution elements of existing client/server distributed computing systems. Wei, Fedak and Cappello published work in 2005 that examined the use of BitTorrent to be the mechanism for distributing data in a computational desktop grid. They followed this work up with replacing the data distribution mechanism in BOINC with BitTorrent (Costa et al 2008)[12].

3.3.2 Peer-to-peer Distributed Computing

There are few demonstrable general purpose distributed computing systems that use peer-to-peer as an underlying architecture. Three representative samples are

12 This work references a paper (Goldsmith 2007) published as a part of the research supporting this thesis.

described here, the Gnutella Processing Unit, JNGI based on the the JXTA framework and P-Grid.

3.3.2.1 GPU

The Gnutella Processing Unit (GPU) is a distributed computing platform that allows users to share processing cycles. It is unique in the fact that along with sharing its name, GPU also shares the paradigm of having nodes provide resources equally, similar to its protocol namesake, without the ability to choose which jobs on the network each node contributes to.

GPU is a SourceForge.net[13] hosted project that is being managed and developed by a group of enthusiasts on the World Wide Web.

3.3.2.2 JXTA / JNGI

JNGI is a framework, built on top of another framework called JXTA which allows users to submit jobs to a network of peers. JXTA, by Sun Microsystems, is a framework created as an attempt to standardise P2P communication between a group of peers. JXTA identifies peers by a unique peer ID and groups of peers are able to form as they participate in various applications on the JXTA network.

JNGI consists of several different entities that are hierarchically arranged and responsible for a number of functions. There are the task dispatcher, repository, code repository, job repository, task repository and then the participating nodes themselves.

3.3.2.3 P-Grid

P-Grid (Aberer 2001) is similar to Pastry (See 3.1.5.1.2) in that it provides a DHT underlying implementation that the overlay is built on top of. What sets it apart is that its focus has been on general application so that it is able to be tasked for more general purpose usage.

At time of writing P-Grid is in limited release to academics looking to investigate peer-to-peer and grid systems.

3.3.3 Botnets

A Botnet is a colloquialism for a massively distributed computing system that typically engages in malfeasance on a broad scale automatically utilising computer systems that are usually unaware that they are even a part of the network. Studies presented at a recent conference (HotBots07) have suggested that some of the most distributed, peer-to-peer computing systems in known usage, capable of multiple tasks and dynamic control, are potentially Botnets.

Whilst hardly a peer-reviewed system in its own right, it would be remiss not to mention potentially the largest deployed massively parallel peer-to-peer computing system in existence. Research into quantifying (Grizzard et al 2007) and reverse-engineering (Chiang & Lloyd 2007) some of these systems is now an emerging field.

13 SourceForge.net is a web based project management and source code management tool. It is a serves as a development tool for distributed collaboration and also as a central location for finding new and existing projects.

3.4 Benchmarking and Evaluation

Several benchmarking suites exist specifically for the field of grid or distributed computing. The NAS Grid Benchmarks provide a detailed specification for a group of problems that can be solved by a grid in order to gain some benchmarks and insights. These grew out of the earlier NAS Parallel benchmarks (NPB), which represent a suite of problems that were typically encountered by NASA when investigating computational aerodynamics. Earlier work, largely based on benchmarking discrete computer systems such as LINPACK (explained below), have also been retrofitted and applied to distributed computing systems. Specialist grid benchmarking suites have also emerged such as Grid Bench that aim to take measurements across the spectrum of a grid's operation.

Whilst these benchmarks exist, few examples of comparative results with systems that are actually in use today are available to the researcher. This lessens the understanding of existing approaches and makes it difficult to assess the contribution of new systems as they emerge.

3.4.1 Whetstone and Dhrystone

Whetstone was one of the first general purpose, open benchmarks available (Curnow & Wichman 1976). It is a synthetic benchmark developed to test the floating point performance of an arbitrary machine whilst avoiding many known compiler optimisations. It produces a measure known as Whetstone Instructions per Second. Its name comes from the town in which the benchmark was developed. Many historical and modern machine's performance figures are available online (Longbottom 2008).

A play on the name of the Whetstone benchmark, Dhrystone is a synthetic benchmark developed to test the integer performance of a computing system in much the same way the whetstone measures floating point performance (Weicker 1984). It to was derived from a statistical analysis of computer operations and primarily tests the arithmetic and logical performance of a CPU.

3.4.2 Netperf

Netperf is a network performance measurement benchmark tool which focusses on the measurement of Ethernet performance by sending messages of various blocksizes from one host to another. This allows for measurements of latency, round trip time, saturation points and so on (Netperf 2009).

3.4.3 LINPACK and LAPACK

LINPACK is a library of routines for the solving of linear equations and least-squares problems (Dongarra et al 2003). It was developed during the late 1970s and early 1980s for use on supercomputers. LAPACK is an updated version of LINPACK redesigned to take advantage of vector based, shared memory supercomputers. The LINPACK benchmark, derived from the usage of LINPACK itself, can measure a computer systems power by solving a system of linear equations. This provides a useful measure of floating point performance since solving dense linear equations is a common task for supercomputers for a variety of engineering problems.

3.4.5 NAS Grid Benchmarks

The NAS Grid Benchmarks (NAS stands for Numerical Aerodynamic Simulation) are the logical succession from the earlier NAS Parallel Benchmarks which were developed at NASA as a set of common calculations that they often do. The original benchmarks were published as "pencil and paper" algorithms where the calculations were explained and input data and corresponding expected output were given (Bailey et al 1991). Example implementations came later with NAS 2. These were MPI based implementations that could be executed on any machine with an MPI implementation. NAS Parallel Benchmark 3 (NPB 3) was a further evolution with an implementation in OpenMP (a shared memory parallel programming library), high performance Fortran and Java. The latest release is GridNPB3 or the NAS Grid Benchmarks. These comprise of four benchmarks that are suitable to be run on a computing grid. There is an implementation available in both Fortran and Java using the Globus toolkit as a grid platform. A protocol for the usage of the NAS Grid Benchmarks is still being developed, at time of writing, so no comparative information between grid performance is available.

3.4.6 Peer-to-peer Benchmark Work

As alluded to in 2.3.3, peer-to-peer specific benchmarks are still very much in their early stages. Rhea et al (2003) present two benchmarks, *find_owner* and *locate*, as two common activities that DHT based peer-to-peer system rely on. Jinyang Li's PhD thesis (2005) presents a performance versus cost analysis of several DHT designs. Her performance versus cost metric (PVC), under simulation when graphed, produces a convex hull as the cost (bandwidth) varies against the performance (mean lookup latency). These convex hulls represent a best achievable performance versus cost combination for a DHT under given parameters.

3.5 Chapter Summary

This chapter has briefly reviewed the most relevant examples of projects, based on the fields discussed in chapter two, that provide background for the remainder of this thesis. This provided and discussed examples of peer-to-peer network implementations, distributed computing and benchmarking and evaluation. Also described was a new taxonomy for peer-to-peer network implementations that allows existing and future designs to be categorised as one or more of the following: *Store and Forward, Centralised, Decentralised, Structured, Unstructured* and *Hybrid*.

Chapter Four

4.0 WAN-DC: A New Framework for the Comparison of Wide Area Network Distributed Computing Systems

The emerging field of peer-to-peer wide area network distributed computing lacks a practical framework to provide a meaningful comparison between competing distributed computing systems and approaches. How does a peer-to-peer approach compare to a client-server approach? Does Condor solve an embarrassingly parallel problem quicker than BOINC? These questions become crucial when looking to compare and contrast the performance and usability of a new distributed computing or grid system with existing offerings.

This work aims to take the first steps in providing a concise, general-purpose benchmarking suite suitable for emerging wide area network distributed computing systems.

4.1 Motivation

This thesis represents part of the convergence between peer-to-peer systems and distributed computing. This convergence is something that has been described in the literature as being as certain "as death and taxes" (Iamnitchi and Foster 2003), yet still few operational systems exist which demonstrate this apparent foregone conclusion. The result of this investigation thus far has been the development of a hybrid peer-to-peer system called CompTorrent (Goldsmith 2007) and the development of the thesis that a peer-to-peer distributed computing platform should provide, in many cases, comparable performance to a client-server distributed computing system such as BOINC or Condor. After the system was developed, and even with the knowledge of distributed computing benchmarks that have already been mentioned (See 3.4), there was a real lack of published results of comparable systems. This has led to an expansion of the work to include a wider field of experimentation than what was originally intended.

This is not to say that some comparative work has not already been done between the various grid systems, just not as much as one might expect given the abundance of systems available. Radić and Imamagić (2004) have provided results on the performance of several job management systems and provided comparative results between the Sun Grid Engine, Torque and Condor. The NAS Parallel benchmarks also included some sample results based on earlier parallel machines (Bailey et al 1991). So called "stat races" are also a prominent feature in online, volunteer distributed computing. Many project groups doing processing will provide online statistics regarding total number of users, CPU years, aggregate memory (in terms of RAM) and so on. Many of these statistics are from BOINC projects with a mix of other technologies such as distributed.net, Apple's X-Grid and others (Volunteer at Home 2007). These provide an ad hoc method of comparing cluster size and performance between projects competing for volunteer resources.

Other comparative work has looked more at differences in different cluster hardware, rather than the underlying protocols themselves. Chang et al (2004) have

looked at the difference between CISC and RISC hardware when executing the NAS Parallel Benchmarks. Boklund et al (2005) have produced some interesting work comparing the different performance of cluster hardware over time.

Little work appears to have been done on performance measurement and comparison between distributed computing platforms that exist over wide area networks. This benchmark is a first attempt at addressing this need to compare these systems in a quick and practical way.

4.2 Performance Metrics

This section describes the algorithms selected to be a part of the benchmark and the reasons behind their selection. There have been two primary goals that this has set out to achieve. Firstly, and similarly to NASA during the selection of their original grid benchmark, a range of algorithms have been chosen that satisfy the current requirements that this research has for distributed processing. In this case, a benchmark is needed that can be applied quickly and that is able to be computed by an arbitrary distributed computing system. The benchmark should produce results without needing to focus on the implementation of the benchmark itself.

Dedicated clusters, in many cases, can be measured using benchmarks derived from parallel machines. However, wide area distributed computing, peer-to-peer and grass-roots volunteer computing, where participating nodes are highly transient, does provide numerous new challenges. These challenges are not based in performance metrics alone: the voluntary and participatory nature of these new designs require a benchmark to consider some of the more qualitative aspects of the exercise as computing efficiency alone can be less important, to the outcome of the project, than the ability to attract participants. The costs of joining and the effort required to start and host a distributed computing exercise can have a large effect on the success or outcome of a modern project as it tries to attract participating nodes. Thus it is the case that this benchmark also introduces an evaluation criteria for distributed computing systems that can serve as a basis for evaluation, comparison and selection.

The following principles have been followed in the creation of this benchmark:

- Load should be able to be varied without changing the algorithm
- Data used should be freely available and easily verifiable
- The benchmark itself contains both practical applications and synthetic mixes. That is, applications that have a practical use, such as processing video, as well as purely synthetic tests that are only for the sake of benchmarking such as performing no work at all
- The benchmark is easily extensible with room for extra algorithms to be added in

4.3 Computing Benchmark

Independently, or Embarrassingly, parallel problems (see Chapter Two) represent algorithms that have a high level of natural parallelism. That is, they can be trivially divided amongst multiple processing elements for them to work independently towards a solution.

Four separate sections here are included to test the various parts of a typical

distributed computing system. These sections consist of baseline tests, to gain a comparative measure against other existing and historical machines then metric categories of *processing intensive*, *mix* and *responsiveness/overhead.*

4.3.1 Baseline

A number of existing benchmarks have been selected to provide a measure of the processing power of the computing equipment being measured. This can be used to compare a distributed computing design to an earlier supercomputer approach where often Whetstone or LAPACK figures are likely to exist.

The Netperf benchmark is included to allow for the comparison of network performance between different machines. Network performance directly relates to the distributed computing system's ability to perform efficiently. It can also detect unusual circumstances that exist on a network that would otherwise have unfairly affected the score of a system's performance (Gropp & Sterling 2005).

4.3.1.1 LAPACK (synthetic)

Test Description	Obtain wall clock (time elapsed measured from start to finish), CPU time and operations per second to complete the LINPACK benchmark.
Test Aim	To produce comparative performance figures using a well known and accepted benchmark.
Input Data	A freely, Internet-available dataset of a suitable size for the test. Matrix Market is a recommended source repository for data.
Output Data	LINPACK output. Wall clock time to complete the benchmark.
Method	Application should be run from 1 node to all node machines with a reasonable number of steps to meet a stated confidence interval (95% suggested). Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin. The overall experiment should be repeated for each distributed computing application, on unchanged hardware, after a complete reboot or equivalent cache clearing.

Table 3: The LAPACK experiment in detail in the WAN-DC benchmark.

4.3.1.2 Whetstone & Dhrystone (theoretical maximum)

Whetstone is a classic synthetic benchmark consisting of an instruction mix derived from a statistical analysis of scientific applications running on an early computer system from the 1960s. Whilst the results from Whetstone in Mflops/sec are unlikely to be similar to LINPACK (such is the nature of benchmarks), it does provide a good idea of the “overall size” of a cluster. This is useful for comparison with the vast number of machines that have had statistics gathered with Whetstone over a considerable period of time.

Test Description	Obtain figures from the Whetstone benchmark
Test Aim	An attempt to gain a best estimate of floating point numerical performance of the cluster hardware. This is to serve as a working theoretical maximum for what a distributed computing system could hope to achieve as overheads approach nil.
Input Data	Nil
Output Data	Whetstone benchmark output for each worker node in the cluster (produced by application with results to be added together by tester).
Method	Application should be run from 1 node to all node machines with a reasonable number of steps to meet a stated confidence interval (95% suggested). Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin.

Table 4: The Whetstone and Dhrystone experiment in detail in the WAN-DC benchmark.

4.3.1.3 Netperf / Netpipe (theoretical maximum)

Netperf is a network latency and throughput benchmark.

Test Description	Perform the netperf benchmark on the cluster system.
Test Aim	An attempt to gain a best estimate of network performance of the cluster hardware. This is to serve as a working theoretical maximum for what a distributed computing system could hope to achieve in terms of network performance. This is used as a reference point for the earlier input and output benchmarks in order to isolate network from protocol overhead.
Input Data	Nil
Output Data	Netperf benchmark results.
Method	Application should be run from 1 node to all node machines with the same number of intermediary steps as used in the other benchmarks.

Table 5: The Netperf experiment in detail in the WAN-DC benchmark.

4.3.2 Processing Intensive

This benchmark application is designed to evaluate the processing power of a cluster.

4.3.2.1 POV-Ray Benchmark (application)

POV-Ray (Persistence of Vision Raytracer) is a ray tracing program available for a wide variety of computer platforms. It has an accepted benchmark scene which, when rendered, allows for the comparison of different systems ability.

Test Description	Obtain wall clock and CPU time to complete the POV-Ray benchmark.
Test Aim	To gain an understanding of numerical performance of the system, with minimal network and disk usage, using an application benchmark that can be executed identically between different known distributed computing systems.
Input Data	benchmark.pov benchmark.ini
Output Data	Wall clock time to complete the benchmark from start to finish.

Method	Application should be run from 1 node to all node machines with a reasonable number of steps to meet a stated confidence interval (95% suggested). Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin. The overall experiment should be repeated for each distributed computing application, on unchanged hardware, after a complete reboot or equivalent cache clearing.

Table 6: The POV-Ray experiment in detail in the WAN-DC benchmark.

4.3.3 Mix

These Mix benchmarks are designed to compare systems performing a mix of network and disk I/O whilst also performing processing work.

4.3.3.1 Transcode (application)

Transcode is an open source application that enables the conversion of one video format to another. For the purpose of benchmarking, video reprocessing is both a disk, processor and network intensive process.

Test Description	Obtain bandwidth figures (Mbytes/sec) and wall clock time to distribute data over the cluster using the subject protocol. Obtain wall clock and CPU time to complete the video processing conversion from transcode. Obtain overall clock time to complete the video processing from start to finish.
Test Aim	To gain an understanding of the overall performance of the cluster system with an application benchmark that involves intensive processing, network/ disk IO and large input and output datasets.
Input Data	A freely, Internet-available dataset of a suitable size for the test. Suggested data element sizes ranging from 1Mb, 2Mb, 5Mb, 10Mb, 100Mb, 1Gb. Care should be taken to ensure that data element size ranges both under and over the available memory per node to test the effect of changes to input/output requirements on each participating node. The Internet Archive Moving Image Archive is a suggested repository for source data.
Output Data	Transcode timing output for each worker node in the cluster (produced by application with results to be tabulated by the tester). Wall clock time to complete the benchmark from start to finish. The resultant converted video files.
Method	Application should be run from 1 node to all node machines with a reasonable number of steps to meet a stated confidence interval (95% suggested) for each data element size. Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin.

Table 7: The Transcode experiment in detail in the WAN-DC benchmark.

4.3.3.2 Mandelbrot (synthetic)

The Mandelbrot Set is a fractal for which a large number of implementations are available and has been a popular choice for benchmarking due to its large, complicated output from relatively simple code (Vaughan & Brookes 1989).

The following is an excerpt of the algorithm used as derived from Emmanuel Dumas' (2001) implementation.

```
int mandelpoint(NUMBER z0_r, NUMBER z0_i, int nmax) {

        int cp=0;
        NUMBER zn_r,zn_i,a,b;

        zn_r=z0_r;
        zn_i=z0_i;
        a=zn_r*zn_r;
        b=zn_i*zn_i;

        while ( (cp<nmax) &&( (a+b) < SEUIL ) ) {
              zn_i=2.0*zn_r*zn_i+z0_i;
              zn_r=a-b+z0_r;
        a=zn_r*zn_r;
              b=zn_i*zn_i;
        cp++;
        }
        return cp;
}

void mandelbrot(const NUMBER xmin,
                const NUMBER xmax,
                const NUMBER ymin,
                const NUMBER ymax,
                const int startx,
                const int starty,
                const int lx,
                const int ly,
                const int nmax,
                const int ximgmax,
                const int yimgmax,
                int *res) {

        int x,y;

        for(x=startx;x<startx+lx;x++) {
                for(y=starty;y<starty+ly;y++) {
                        NUMBER z0_r,z0_i;
                        // calcul de z0
                        z0_r=xmin+(xmax-xmin)*x/ximgmax;
                        z0_i=ymin+(ymax-ymin)*y/yimgmax;
                        res[(x - startx)+(y-starty)*lx]=mandelpoint(z0_r,z0_i,nmax);
        }
        }
}
```

Test Description	Obtain overall wall clock time to complete the Mandelbrot set.
Test Aim	To gain an understanding of the overall performance of the cluster system with an application benchmark that involves intensive processing, network/ disk IO and large output datasets.
Input Data	A set of files each representing the region of the Mandelbrot set to calculate. There are three separate parameter sets to this bench mark. 1. The overall set size being generated. 2. The number of segments the set is to be split into to be computed in parallel. 3. The region of the Mandelbrot set to be calculated.

	Depending on items 2 and 3, each work unit can be either relatively uniform or provide a mix ranging from intensive computation to near trivial. Two datasets are provided, Mandelbrot_A and Mandelbrot_B, which illustrate each option.
Output Data	Image files of the Mandelbrot set. The wall clock time to produce the set.
Method	Application should be run from 1 node to all node machines with a reasonable number of steps to meet a stated confidence interval (95% suggested) for each data element size and Mandelbrot region (Mandelbrot_A and Mandelbrot_B). By default, vertical "striped" regions are to be calculated with any deviation (such as square regions instead) being clearly noted. Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin.

Table 8: The Mandelbrot experiment in detail in the WAN-DC benchmark.

4.3.4 Responsiveness / Overhead

The responsiveness and overhead benchmarks are designed to load the system with as much "non-work" as possible. This "non-work" has a known amount of time to complete in terms of clock time per work unit. Thus the difference between the aggregate time of the work units and the actual time the system takes to complete them, when compared between systems, can give the experimenter an average indication of the throughput capabilities of each system as well as a measure of overhead.

$$Overhead = \frac{1}{n} . (\sum_{i=1}^{n} (t(onesec)_i - k_i))$$

Where:

n is the number of samples taken
k is the ideal clock time of the computation
$t(onesec)$ is the clock time of the one second computation

4.3.4.1 One second (synthetic)

One second is a small application which will loop for as near to one second as possible.

A C implementation is as follows:

```
// get start time

timeval tim;
gettimeofday(&tim, NULL);
double t1=tim.tv_sec+(tim.tv_usec/1000000.0);
double t2 = 0.0;

unsigned long loops = 0;

do {
        gettimeofday(&tim, NULL);
        t2=tim.tv_sec+(tim.tv_usec/1000000.0);
        ++loops;
} while( (t2 - t1) < 1.0);

printf("%.6lf seconds elapsed. %d loops.\n", t2-t1, loops);
```

Test Description	Load the system with work units with a known execution time.
Test Aim	To see the difference between competing systems when executing for a known period of time. How long does it actually take to compute a known period of work?
Input Data	The length of time to compute for each work unit. Suggested lengths include: *1 sec, 10 secs, 100 secs, 1000 secs, 2000, 3000, etc.*
Output Data	The wall clock time in seconds.
Method	Application should be run from 1 node to all node machines with a reasonable number of steps to meet a stated confidence interval (95% suggested) for each work unit length. Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin.

Table 9: The One Second experiment in detail in the WAN-DC benchmark.

4.3.4.2 Mean work unit (synthetic derived from application)

Clock time difference between work units being completed over the course of computation.

Test Description	Obtain wall clock times for the difference between completed work units.
Test Aim	To investigate the comparative lag that occurs between distributed computing systems. This is especially important for comparing emerging systems that rely on a decentralised approach where overhead may effect the ongoing performance of the system.
Input Data	Data can be gathered from the results of other benchmark experiments from both a server or node perspective. That is, the time taken for each completed work unit to be received. Or, the time between starting a new worker node and actually receiving data to process (where data is available).
Output Data	The wall clock time in seconds.
Method	Application should be run from 1 node to all node machines with a reasonable number of steps to meet a stated confidence interval (95% suggested) for each work unit length. Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin. Care should be taken to ensure that the test is fair and that all systems being compared are set to begin computation as soon as possible and that network load is comparable between tests.

Table 10: The Mean Work Unit experiment in detail in the WAN-DC benchmark.

4.3.4.3 No work (theoretical maximum)

The *No Work* test results in the number of empty work units that can be completed in a period of time.

A implementation in C, to literally do nothing, is as follows:

```
int main (int argc, char * const argv[]) {

        return 0;
}
```

Test Description	Obtain a theoretical maximum for the number of work units a system can process as the processing load of the work unit approaches zero.
Test Aim	To investigate the maximum number of work units that can be issued and processed in a given amount of time.
Input Data	None.
Output Data	The number of completed work units in a given period of time.
Method	Application should be run from 1 node to all node machines with a reasonable number of steps to meet a stated confidence interval (95% suggested) for each work unit length. Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin. Care should be taken to ensure that the test is fair and that all systems being compared are set to begin computation as soon as possible and that network load is comparable between tests.

Table 11: The No Work experiment in detail in the WAN-DC benchmark.

4.4 Qualitative Issues

This section is intended to serve as both a selection criterion for distributed computing systems as well as a part of a framework to evaluate and compare them. It has been developed from a literature review of available distributed computing systems, grid computing and peer-to-peer computing areas. These areas continue to relate and converge with each other (Foster & Iamnitchi 2003) and now require a consistent benchmark that has useful application across all three areas.

There has been some substantial earlier work in some of these areas. In resource management systems there exists a comprehensive taxonomy and survey of existing systems (Krauter, Buyya & Maheswaran 2002). Object modelling, within distributed computing, has been investigated with a view towards classification and comparison (Bakker, Kuz & van Steen 1997).

This framework draws on this earlier work, and on recent developments in the direction of distributed computing, to produce a high-level comparative framework.

4.4.1 General Approach & Design

In order to categorise the system being examined, it is necessary to examine the approach taken in order to complete a distributed computing exercise, the overlay network paradigm chosen, how the network is organised, whether the focus is on a task or a service and the requirements necessary to participate.

4.4.1.1 Approach

How is the distributed system organised? The first consideration should be whether the system comprises of a group of general machines or consists of specialised hardware. If the system is general hardware, what is the approach taken to provide

computing services? Is it a decentralised swarm of loosely cooperating machines, a distributed cluster of machines, is it a grid providing services or is it a cluster of machines pooled between different geographic areas?

Specialised Hardware	General Purpose Hardware
Parallel supercomputer	Swarm
	Grid
	Cluster
	Ad Hoc

Table 12: The qualitative approach category in the WAN-DC benchmark.

4.4.1.2 Node Organisation (client / server, peer to peer)

Distributed computing systems require nodes to organise themselves so that communication can occur so that jobs can be allocated and results returned. Client/server, peer-to-peer and a hybrid approach are the three known methods used to achieve this. Other methods will undoubtedly emerge and should be also considered to allow the categorisation of new systems.

Node Organisation
Client/server
Peer-to-peer
Hybrid
Other

Table 13: The qualitative node organisation category in the WAN-DC benchmark.

4.4.1.3 Network Topology

In many node organisations (4.4.1.2) different underlying network topologies can be employed for communication between nodes. These can have an effect on the overlayed services.

Network Topology	Description
Fully connected	Each node in the network is connected to every other node in the network.
Ring (or list)	Nodes are arranged into a ring or list with a half or full duplex connection between them.
Tree (or star)	Communication is directed via a strict hierarchy where a root node or central server communicates only with the nodes beneath it and so on. A star configuration is particularly common in client-server systems where a single machine (or group of machines) is connected to nodes around it.
Hypercube	A hypercube is a compromise between a fully connected network and a ring. It aims to produce a minimum number of

	"hops" between nodes to communicate whilst maximising the number of nodes able to participate with a limited number of connections.
Butterfly	A butterfly network, with similar routing abilities to a hypercube, typically has two input and two outputs where combinations of straight, crossed and broadcast paths are possible.
Random	A node joins any other node as it sees fit. There is no deterministic arrangement to how nodes are organised.
Heuristic	Often this would be a network that begins as random and connections change as faults or inefficiencies are discovered in the network.
Other	Other is particularly important as there are many variations and combinations of each one of the topologies described here. Flexibility should be used to describe a variation and refer back to one or more of the high level topologies presented to maintain context.

Table 14: The qualitative network topology category in the WAN-DC benchmark.

4.4.1.4 Application

This section is to determine whether the system is service based or task based. Is the system providing a computing service or is it centred around the computation of a specific task? Indeed, some systems can be configured to do both. The purpose of this section is to clarify this functionality in a distributed computing system.

Application
Task based
Service based
Either

Table 15: The qualitative application category in the WAN-DC benchmark.

4.4.1.5 Requirements & Dedication

What system requirements are required for both hosting a project and participating in one? Are there any special hardware considerations? Does a participating node need to be completely dedicated in order to function? Also, how does the system interact with the operating system and present itself to the user?

Requirements (Host or Node)
Specific hardware
Dedicated machine
Background process
Screensaver / idle cycles only

Table 16: The qualitative requirements category in the WAN-DC benchmark.

4.4.2 Features

Further to the general approach, consideration needs to be given to what features the system offers, the standards it supports, the hardware required, how tasks are managed and its robustness and licensing arrangements.

4.4.2.1 Algorithmic Suitability

Support for algorithm type (embarrassingly parallel, interprocess communication)

Algorithm Support
Interprocess communication
No interprocess communication
Ordered problems

Table 17: The qualitative algorithm support category in the WAN-DC benchmark.

Also, where there are ordered tasks, how is the ordering managed?

Task Order Management
Task graph
Petri Net
Heuristics

Table 18: The qualitative task ordering category in the WAN-DC benchmark.

And where there is support for interprocess communication, by which methods or standards does it occur?

Communication Standard
MPI
PVM
Tuplespace
Other

Table 19: The qualitative communication standards category in the WAN-DC benchmark.

4.4.2.2 Standards Support

What open standards does the system adhere to which enable a deeper comparison between systems?

4.4.2.3 Hardware Support

What hardware does the system support? Is the system able to run on multiple machines and operating systems? If so, which ones? Can the system operate well in a heterogeneous mix of machines and networks and take advantage of individual node's characteristics (such as multi-core processors?). What level of platform independence is there? Graphic processing units (GPU), especially for vector processing, are now potent processors in their own right and are being utilised by various voluntary computing systems.

Hardware Support
Platform Independence
Processing Hardware Support
Heterogeneous Support
Graphics (GPU) Processing support. e.g. High end video cards in modern PCs.

Table 20: The qualitative hardware support category in the WAN-DC benchmark.

4.4.2.4 Task & Resource Management

Resource allocation can be directed by a server or, in some known systems, by nodes choosing which jobs to participate in.

Allocation of Resources / Tasks	**Description**
Job Allocation Control	Server or Node. Are jobs allocated by the server in a server-client fashion? Or can the node choose which work to do from an available pool?
Proportional	Tasks are allocated based on a best fit to node performance and task size.
Market	Tasks and nodes are matched based on a virtual economic market.
None	Nodes are allocated (or can randomly choose) the next task ready for computation (i.e. Next in the queue or graph as determined by the task ordering algorithm)

Table 21: The qualitative job allocation category in the WAN-DC benchmark.

4.4.2.5 Robustness

Robustness	Description
Checkpointing	Support for long task checkpoints, no support (repeat tasks only).
Scalability	Statistical evidence, Observed evidence.
Quality of Service	Types of support being offered or no support.
Churn	Hand over of part completed tasks, task rescheduling, no tolerance for node failure.
Malfeasance	Multiple computation, trust/signing schemes.

Table 22: The qualitative robustness category in the WAN-DC benchmark.

4.4.2.6 Licensing

Licensing	Description
License description	{Proprietary, GPL, Public domain, Other}
Source availability	{Open, Closed}
Governing organisation	Corporation, Non-profit organisation, community, individual.

Table 23: The qualitative licensing category in the WAN-DC benchmark.

4.4.3 Usability

It would be possible with the use of survey and other qualitative research techniques to obtain actual evidence and data for this section, however detailed suggestions for survey or experimental design are left as being beyond the scope of this work.

Usability	Description
Hosting a new project	A originating node or any size. A tracker which can be hosted on the originating node or elsewhere.
Joining an exiting project	Download software and pick a project.
Coding for a new project	None required. Can use existing binaries.

Table 24: The qualitative usability category in the WAN-DC benchmark.

4.4.3.1 Hosting

How much work is needed to begin a project? Are there dedicated requirements for network connectivity or hardware infrastructure?

Hosting a project	
Software Perspective	*Hardware Perspective*
Custom software required	General commodity hardware
Dedicated operating system	Specific hardware

Table 25: The qualitative hosting category in the WAN-DC benchmark.

4.4.3.2 Joining

What effort is required to join a running project? Is downloading and installing software required?

Joining an existing project as a participant
Custom software required
Attaching through another application (i.e. Web Browser, Java Web Start)
Dedicated operating system

Table 26: The qualitative joining category in the WAN-DC benchmark.

4.4.3.3 Coding

Coding for hosting a new project
Support for existing binaries.
Libraries need to be compiled in.

Table 27: The qualitative coding category in the WAN-DC benchmark.

4.4.3.4 Support

What support is available for managing the system?

Support
Commercial support
Community support
Books
Developer support

Table 28: The qualitative support category in the WAN-DC benchmark.

4.4.4 Incentives

How are nodes attracted to participate in a project? General techniques such as publicity, etc. are discussed by projects such as BOINC (BOINC 2007). What features are provided by the software or system itself to encourage nodes to participate in the project?

Incentives	Description
Data access	Access to the source or computed data set.
Financial through statistics	Reliable statistical information of work units completed which can then form the basic of an actual economic system where processing work can be billed to the project owner and worker nodes can be paid.
Fame through statistics	Reliable statistical information of work units completed which can be published to encourage new nodes to join and existing nodes to remain to hold or improve current leader board placement.

Table 29: The qualitative incentives category in the WAN-DC benchmark.

4.5 Discussion of the WAN-DC benchmark

Like the NAS parallel benchmarks, these have grown out of a personal requirement rather than an attempt to be all things to all benchmarking systems. This is obvious given that the operational characteristics (Section 4.3) cover only independently parallel problems or that malfeasance, of which some aspects are possible to measure, is not given here. As such, detailed discussion of suitable further work, including support for algorithms that require interprocess communication, is discussed in Chapter Nine as a part of future work.

4.6 Chapter Summary

This chapter has presented the WAN-DC benchmark. It is a benchmark suitable for wide area network distributed computing systems that evaluates both operational characteristics as well as a comparative framework to investigate the qualitative aspects of the system. It is the aim of this benchmark to provide some coverage across the previously separate areas of distributed computing and peer-to-peer networks. This benchmark is used in the next chapter to evaluate two client-server distributed computing systems and later in chapter eight to evaluate a hybrid peer-to-peer system.

Chapter Five

5.0 A Comparative Evaluation of Condor and BOINC Using the WAN-DC Benchmark

This chapter applies the developed benchmark presented in the previous chapter to compare two existing wide area network distributed computing systems. This comparison forms the foundation for the CompTorrent evaluation in chapter eight. All comparisons have occurred on the same hardware and network under controlled conditions to produce a fair evaluation of the different platforms. These controlled conditions come at the expense of not comparing over a wider area network than what was employed. However, a known network/hardware platform was considered more important to compare the new approach with the existing approaches in a meaningful way over what amounts to increases in latency, reduction in bandwidth and added issues of reliability and heterogeneity of network connection and processor. Further work required in this area is later discussed in section 9.2.1.3.

5.1 Test Platforms

BOINC (See 3.2.1.1) and Condor (See 3.2.1.2) were selected in regards to their popularity, ubiquity and feature set. Both are widely used scientifically and differ in their usage and implementation.

The focus of these experiments was a fair comparison to gauge how these systems compare with each other and then later with CompTorrent. All reasonable attempts were made to have similar operating characteristics between the two platforms when experimentation occurred. Communication rates with servers, timeouts, work unit scheduling characteristics and job allocation settings were made as equivalent as possible between the two platforms. This was done to expose the underlying approach whilst not disadvantaging either platform by nature of its operational settings.

5.2 Test Environment

A cluster of 16 Pentium 3 class machines. Each with 800Mhz (approx) processors and all having 256Mb RAM and running Linux (kernel 2.6.12). Each machine was connected to the same network segment on a 100Mbps switched Ethernet network. Section 5.4.1 shows individual cluster machine and network performance.

A 17th machine was employed as the master server for both BOINC and Condor. It was also the same class of machine as each cluster node with the exception of RAM size with it possessing 512Mb in total.

The method for setting up these experiments on BOINC and Condor is described in detail in Appendix C.

5.3 Test Datasets

There are several datasets that are used for this chapter. For the Transcode experiment an arbitrary public domain movie, in MPEG2 format was chosen for

conversion into the AVI format. This video file used here was Charlie Chaplin's "Cruel Cruel Love" broken into 100 pieces of around 1Mb each. The POV-Ray experiment uses the POV-Ray benchmark dataset as included with the POV-Ray application with each work unit broken into 64 horizontal slices of equal height and width. Mandelbrot is a classical rendering of the Mandelbrot set with the overall job broken down into 160 work units to provide at least 10 units of work per machine in our sample cluster.

See Appendix E for a detailed description of the test datasets.

5.4 Results

BOINC and Condor displayed different results when running identical algorithms, with the same data on the same hardware and network platform. These results are given for Condor in section 5.4.2 and then BOINC in section 5.4.3. This chapter concludes with a comparative summary and discussion in section 5.5.

5.4.1 Baseline

Here presented are results for the baseline that forms a part of the WAN-DC benchmark as described in 4.3.1.

5.4.1.1 LAPACK

Benchmark	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
LINPACK Rolled Double Precision	239.11	219.96	220.04	220.61	220.11	220.19	220.13	219.95
LINPACK Unrolled Double Precision	313.32	280.43	280.57	288.9	280.63	280.64	280.45	280.31
LINPACK Rolled Single Precision	179.6	173.27	173.08	173.25	173.21	173.28	173.25	173.09
LINPACK Unrolled Single Precision	194.89	185.98	186.04	186.34	186.04	186.05	186.05	186.01
Benchmark	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
LINPACK Rolled Double Precision	220.17	219.69	220.2	239.15	220.08	220.16	220.14	220.18
LINPACK Unrolled Double Precision	280.41	280.07	280.57	313.2	280.54	280.62	278.4	280.64
LINPACK Rolled Single Precision	173.24	173.1	173.22	179.72	173.27	173.22	173.21	173.29
LINPACK Unrolled Single Precision	186.09	185.85	186.08	194.94	186.07	186.03	186.06	186.06

Table 30: Individual cluster machine performance for LAPACK

5.4.1.2 Whetstone & Dhrystone

Benchmark	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dhrystone VAX MIPS	922.14	850.88	851.01	855.74	856.13	855.74	851.01	855.48
Whetstone MWIPS	734.7	678.15	678.14	678.07	678.2	678.14	678.15	677.61
Benchmark	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Dhrystone VAX MIPS	826.06	849.48	851.26	922.29	851.13	856	855.87	851.13
Whetstone MWIPS	678.13	676.74	678.14	735.1	678.17	678.14	678.17	678.17

Table 31: Individual cluster machine performance for Whetstone and Dhrystone

5.4.1.3 NetPerf

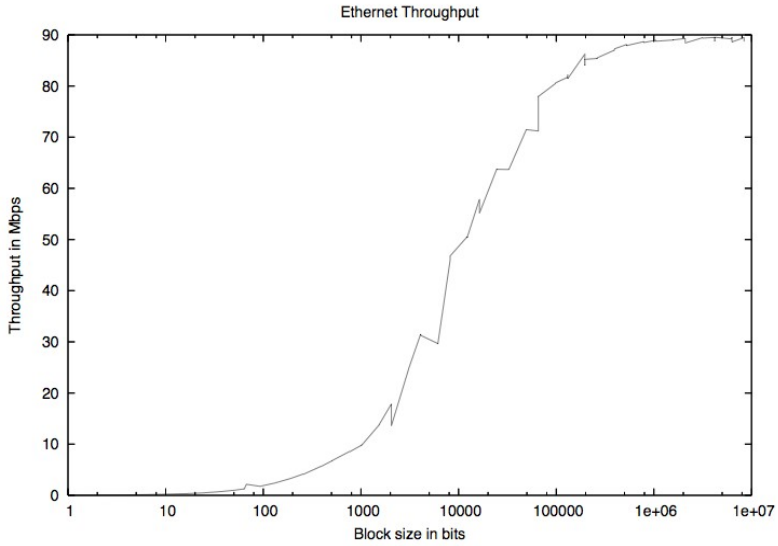


Figure 1: Ethernet throughput of the cluster network.

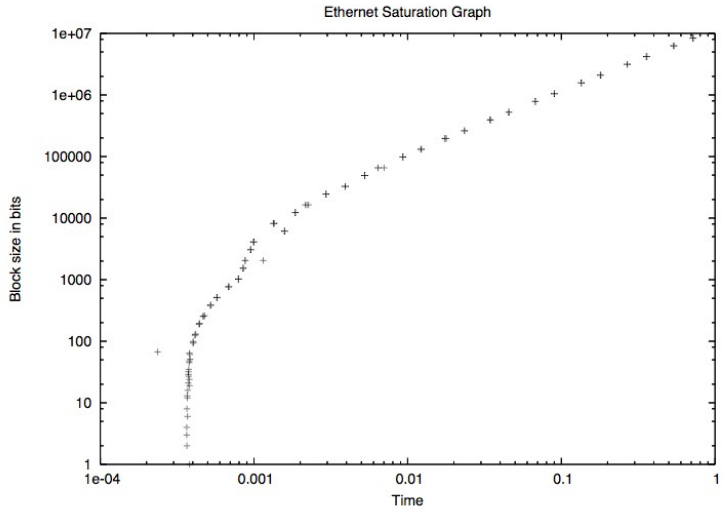


Figure 2: The round trip time of the cluster network with packets of increasing size.

5.4.2 Condor

Appendix C contains detailed instructions and settings for the Condor implementation that forms the basis of these results.

5.4.2.1 POV-Ray

Condor performed well in the POV-Ray experiment as would be expected given its high computation and low network communication requirements. Table 32 shows good speedup efficiency (speedup / number of nodes) but curiously less so with two nodes. Two nodes is where super linear speedup is more likely to occur rather than a slow down. In this case it is simply a single run which performed significantly slower than the rest of the runs. Removing this outlier results in a mean run of 9077.75 with a corresponding rounded speedup of 1.79 and speedup efficiency at 0.895.

Condor POV-Ray				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	16209	19.85	1	1
2	10022.8	2126.67	1.62	0.81
4	4200.2	1.1	3.86	0.97
8	2094	6.24	7.74	0.97
16	1035	2.12	15.66	0.98

Table 32: Condor results for the POV-Ray experiment.

5.4.2.2 Transcode

Some super-linear speedup was observed, quickly faded at 8 nodes with only a marginal improvement at 16. Transcode does have a loading time on these machines in the order of approximately 2 seconds on the data provided and their granularity. This overhead does explain why there is going to be a definite upper bound to the amount of speedup that is going to be observed.

Condor Transcode				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	1162.6	1.82	1	1
2	299	3.94	3.89	1.94
4	238.6	7.3	4.87	1.22
8	215.6	2.61	5.39	0.67
16	213.2	2.77	5.45	0.34

Table 33: Condor results for the Transcode experiment.

5.4.2.3 Mandelbrot

Here Condor has not performed particularly well at all with mean speedup reaching a maximum of 1.86 at 8 machines and performing worse at 16. Mandelbrot, whilst

computationally expensive, requires very little to begin processing and not very much in terms of disk or network input/output. It is likely to be the case that the granularity chosen is particularly poor for Condor's implementation. A comparison with “No Work” in the next section (5.4.2.4) does provide some insight.

Condor Mandelbrot				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	610.8	132.68	1	1
2	420.4	3.51	1.45	0.73
4	333.6	6.35	1.83	0.46
8	329	7.68	1.86	0.23
16	332.2	5.97	1.84	0.12

Table 34: Condor results for the Mandelbrot experiment.

5.4.2.4 No Work

Here illustrated is how Condor performs with almost no work to do. It can be clearly seen that a speedup figure for 2 machines is observed with a maxima at 8 machines and very little change at 16.

This result offers us insight into the results gained in Mandelbrot (5.4.2.3) where it wasn't understood why some speedup wasn't really achieved. This suggests that the Mandelbrot calculation itself was akin to no work under these conditions. There was also considerably more network activity than No Work, with the Mandelbrot experiment, that suggests that the network can be discounted as having an effect on the overall speedup observed in either Mandelbrot or No Work.

Condor No Work				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	345.8	1.92	1	1
2	214	5.48	1.62	0.81
4	213.8	4.15	1.62	0.41
8	211.2	4.09	1.64	0.21
16	209	5.92	1.65	0.1

Table 35: Condor results for the No Work experiment.

5.4.2.5 One Second

Leading on from *No Work* and the observances with *Mandelbrot*, it shows that knowing at what job size the distributed computing system becomes useful is extremely important in the design of a distributed computing project. The One Second experiment is designed to show how the efficiency (expressed as known processing time divided by wall clock time) of a system varies over the size of the work unit.

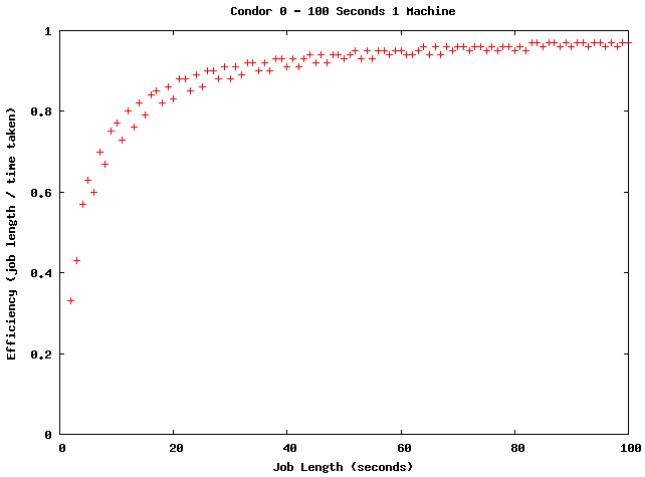


Figure 3: Condor One Second graph for 1 machine.

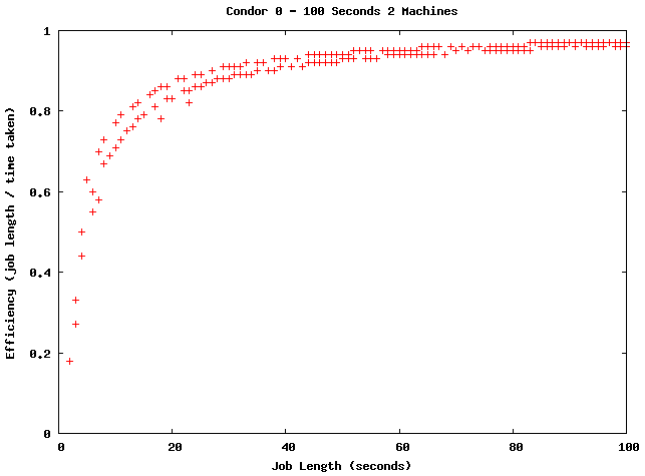


Figure 4: Condor One Second graph for 2 machines.

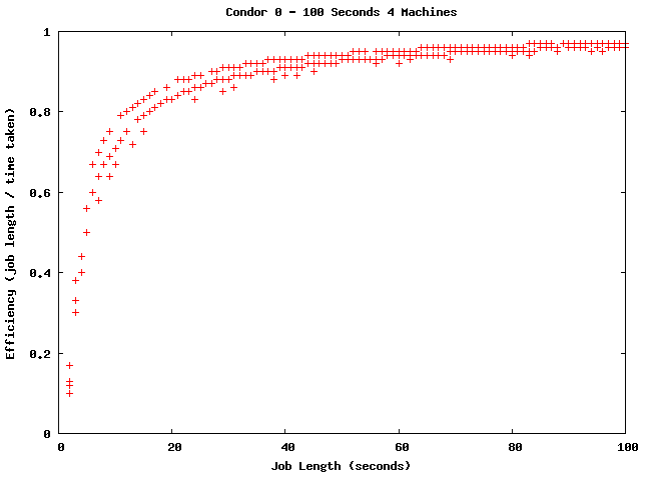


Figure 5: Condor One Second graph for 4 machines.

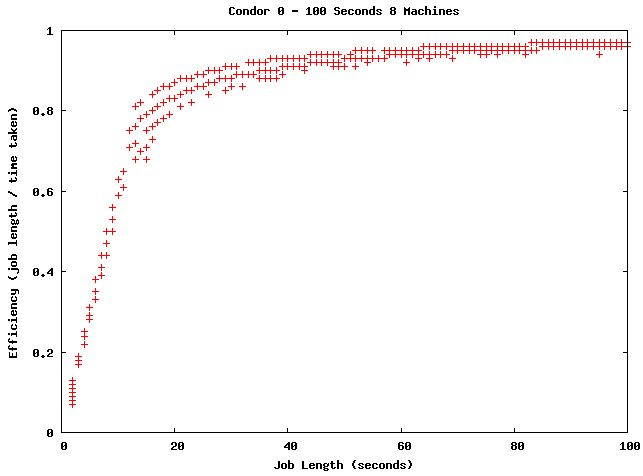


Figure 6: Condor One Second graph for 8 machines.

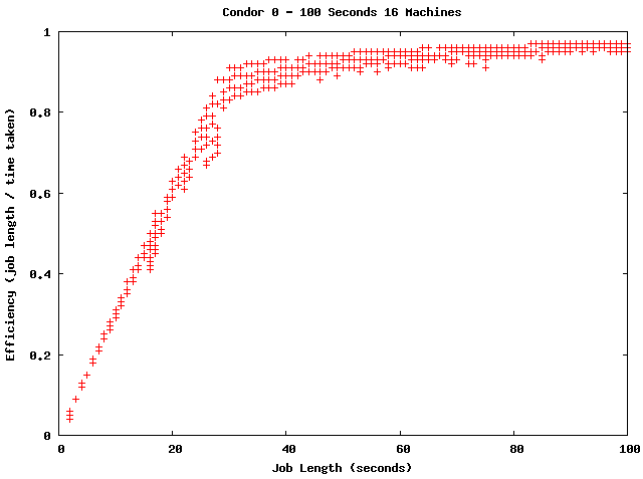


Figure 7: Condor One Second graph for 16 machines.

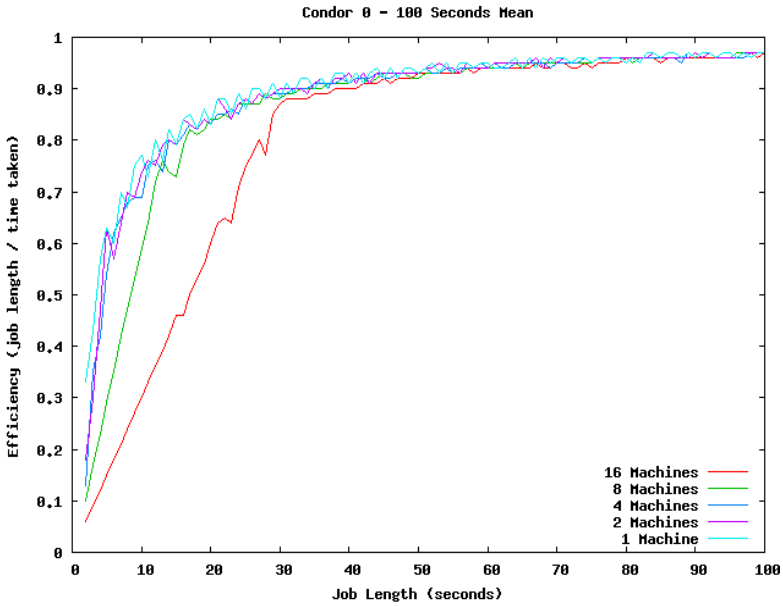


Figure 8: Condor Mean One Second results.

One Second shows , within the bounds of the size of our cluster, that the efficiency of the system converges at a job size of approximately 30 seconds.

5.4.2.6 Mean Work Unit

Table 36 shows the mean work unit results for POV-Ray, Mandelbrot, Transcode and No Work. The most obvious thing it shows is the difference in processing times between the different experiments. Mandelbrot, Transcode and No Work, when compared to POV-Ray, are comparatively light in their processing time per work unit. POV-Ray also displays the largest in the time taken to compute a work unit. This is chiefly due to the nature of the computation itself where the workload varies. The results show that there are cases where there is not a significant difference in run time between 1 and 2 machines.

Care must be taken with varying work unit algorithms like Mandelbrot on a small number of machines. Since Mean Work Unit looks for the average time to get a response with a computed data set, the difference between time taken for each work unit will vary on a single machine as the variance is being driven by the nature of the different work sizes and not the network or the distributed computing system's internal overhead. This is shown obviously with a higher deviation than mean result for Mandelbrot with 1 machine in Table 36.

Experiment	# Machines	Mean (secs)	Standard deviation (secs)
POV-Ray	1	257.43	44.74
	2	159.09	100.42
	4	66.67	58.63
	8	33.24	31.49
	16	16.43	15.88
Mandelbrot	1	3.84	5.67
	2	2.64	0.84
	4	2.10	0.43
	8	2.07	0.43
	16	2.09	0.37
Transcode	1	11.74	1.03
	2	3.02	1.19
	4	2.41	1.2
	8	2.18	1.37
	16	2.15	1.38
No Work	1	3.49	0.23
	2	2.16	0.54
	4	2.16	0.68
	8	2.13	0.54
	16	2.11	0.5

Table 36: Condor results for the Mean Work Unit experiment.

5.4.3 BOINC

5.4.3.1 POV-Ray

BOINC performed extremely well in *POV-Ray*. Table 37 shows good speedup efficiency (speedup / number of nodes) for the first three cluster sizes. Super linear speedup was observed for 8 and 16 node sizes. BOINC takes advantage of both hyper-threading and multi-core processing which explains this super-linearity. For this cluster, hyper-threading was possible (there were no multi-core nodes) and was observed in the BOINC data.

BOINC POV-Ray				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	16846.2	66.82	1	1
2	8640.4	39.39	1.95	0.98
4	4290.6	52.35	3.93	0.98
8	2019.8	21.44	8.34	1.04
16	891.6	32.08	18.89	1.18

Table 37: BOINC results for the POV-Ray experiment.

5.4.3.2 Transcode

Transcode shows increasing speedup across the range of cluster sizes without any super-linear speedup as observed in POV-Ray. The overhead of the network traffic has limited the amount of speedup observed.

BOINC Transcode				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	2619.8	624.68	1	1
2	1647	189.33	1.59	0.8
4	795.2	97.89	3.29	0.82
8	688.2	70.24	3.81	0.48
16	475.6	66.79	5.51	0.34

Table 38: BOINC results for the Transcode experiment.

5.4.3.3 Mandelbrot

Similarly to *Transcode*, *Mandelbrot* shows conventional if uninspiring results with very long runtimes to complete the exercise. Speed up continues to improve over the whole cluster range.

BOINC Mandelbrot				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	2686.6	70.23	1	1
2	1430.4	128.92	1.88	0.94
4	783	58.53	3.43	0.86
8	529.8	135.21	5.07	0.63
16	394.8	181.34	6.8	0.43

Table 39: BOINC results for the Mandelbrot experiment.

5.4.3.4 No Work

BOINC produces a poorer than expected speedup with 4 machines and a maximum speedup with 8 machines. There was little difference between 8 and 16 machines.

BOINC No Work				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	1658.2	178.1	1	1
2	885.2	106.8	1.87	0.94
4	699	164.51	2.37	0.59
8	484	92	3.43	0.43
16	503	65.53	3.3	0.21

Table 40: BOINC results for the No Work experiment.

5.4.3.5 One Second

1,2,4 nodes shows ordered, linear results with very good efficiency (expressed as known processing time divided by wall clock time) ranging from approximately 83 – 101 second job sizes. 1 machine on BOINC shows an unusually high result. This is due to BOINC being able to dispense with its slow and deliberate work unit allocation and request cycle across the network.

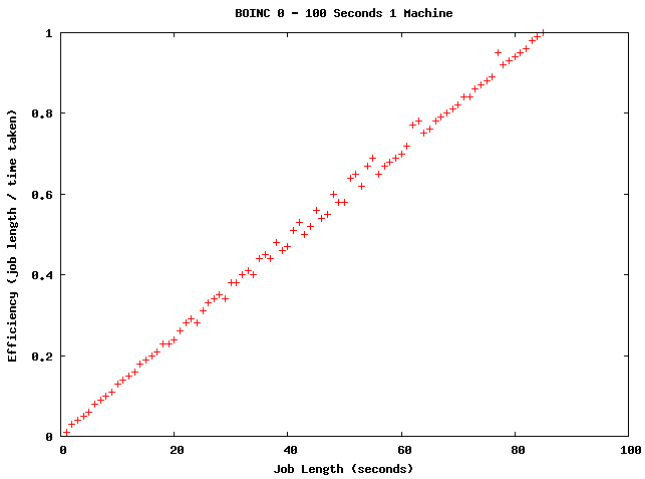


Figure 9: BOINC One Second graph for 1 machine.

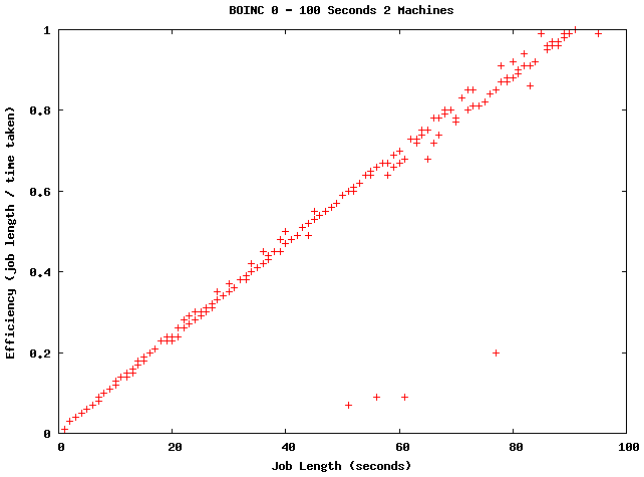


Figure 10: BOINC One Second graph for 2 machines.

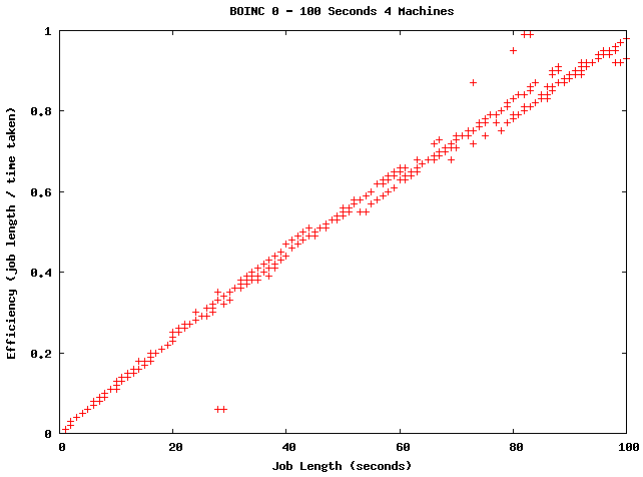


Figure 11: BOINC One Second graph for 4 machines.

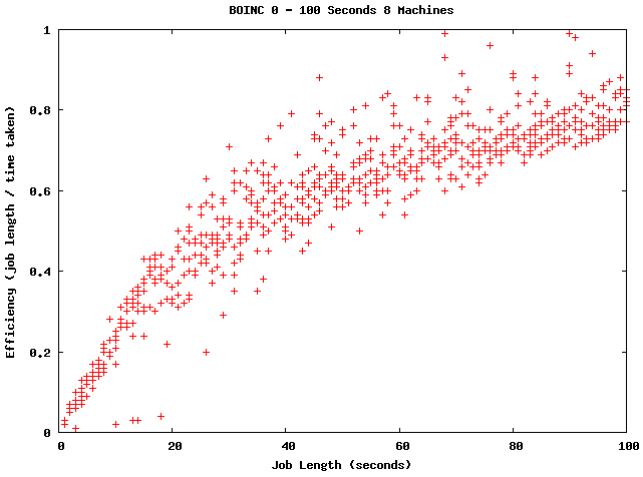


Figure 12: BOINC One Second graph for 8 machines.

However at 8 machines (Figure 12) linearity diverges at 20 seconds with a much higher standard deviation trending towards 100% efficiency at 200 second jobs.

With 16 machines, as expected, the efficiency increases at approximately half the rate of the 8 machine configuration. What is interesting is that the results are much more uniform compared to 8 machines, with more jobs and more nodes.

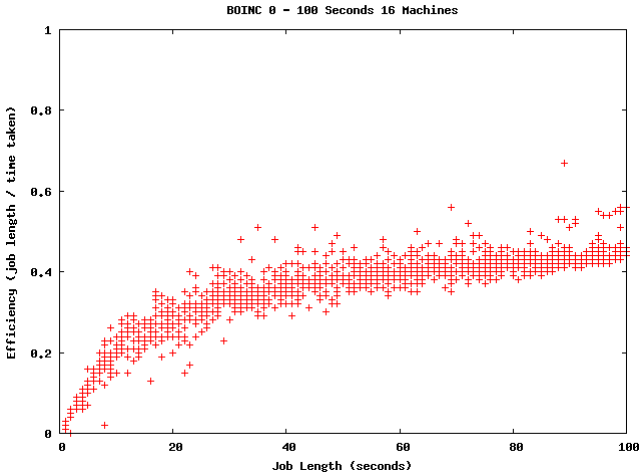


Figure 13: BOINC One Second graph for 16 machines.

The graph in Figure 14 shows all of the previous One Second results as a mean for 1,2,4,8 and 16 node cluster sizes respectively.

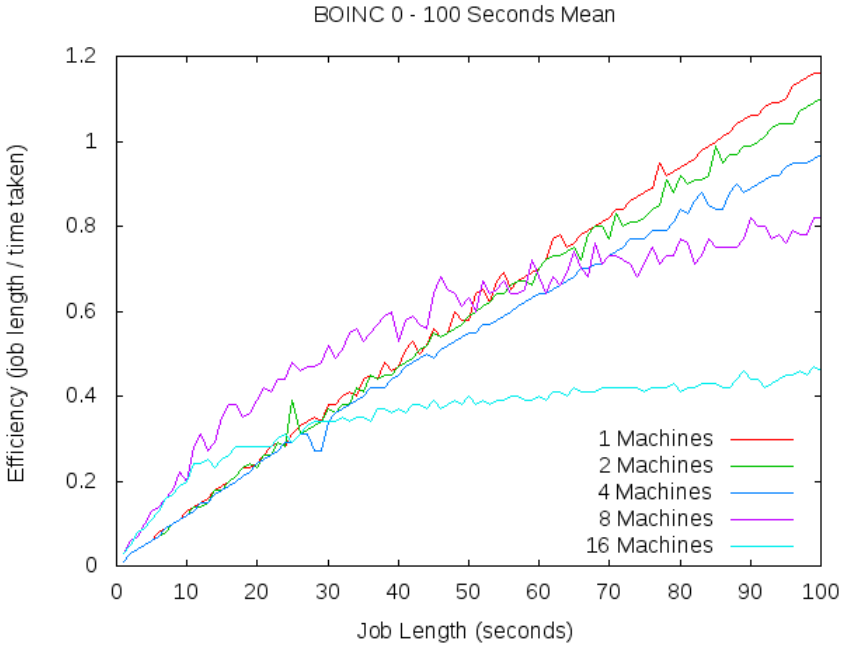


Figure 14: BOINC Mean One Second Results

5.4.3.6 Mean Work Unit

Table 41 shows the mean work unit results for POV-Ray, Mandelbrot, Transcode and No Work. Here deviation is consistently high, as a percentage, of the mean time taken to complete a work unit over the course of all experiments. BOINC will often pause for some time between job requests and allocations. This is due to BOINC's target audience of very large projects with very high numbers of participating nodes where exponential back off on failed requests is required to prevent flooding.

Experiment	# Machines	Mean (secs)	Standard Deviation (secs)
POV-Ray	1	16.75	9.44
	2	8.94	5.97
	4	7.06	10.14
	8	4.89	13.5
	16	5.08	21.14
Mandelbrot	1	16.9	5.16
	2	9	5.96
	4	4.92	3.38
	8	3.33	6.61
	16	2.48	10.73
Transcode	1	26.46	19.31
	2	16.64	12.21
	4	8.03	4.68
	8	6.95	17.94
	16	4.8	10.11
No Work	1	16.75	9.44
	2	8.94	5.97
	4	7.06	10.14
	8	4.89	13.5
	16	5.08	21.14

Table 41: BOINC results for the Mean Work Unit experiment.

5.4.4 Qualitative Evaluation

Table 42 below presents the qualitative results for BOINC and Condor as a comparative guide to the available features of each system.

APPROACH / DESIGN		
	BOINC	***Condor***
Approach	Volunteer / Grid	Cluster / Grid
Node Organisation	Client / Server	Client / Server
Network topology	Star (with super hosts)	Star (with pools)
Application	Task / Grid (with added modules)	Task / Grid (with Condor G)

Requirements & Dedication	Idle cycles or greater	Idle cycles or greater
FEATURES		
	BOINC	***Condor***
Algorithmic Suitability	No interprocess communication	Interprocess communication and ordered problems
Standards supported	None explicitly	MPI, Globus, PVM, Master-Worker (own standard), DRMAA
Hardware / OS support	***Platform Independence:*** Windows, Linux, Mac, Solaris, HPUX, Tru64, AIX, FreeBSD, OpenBSD ***Hardware:*** Intel x86 32/64, AMD 32/64, Power PC 32/64, SPARC, HPPA, Alpha, IA64. Hyperthreading and multi-core support ***GPU Crunching:*** Application based. Evidence of some projects using some specific GPU hardware. ***Heterogeneous:*** Can target specific hardware or any hardware above.	***Platform Independence:*** Windows, Linux, Mac, Solaris, HPUX, AIX ***Hardware:*** As supported by the above operating systems. ***GPU Crunching:*** General support not at present but it is an intention. ***Heterogeneous:*** Can target specific hardware or any hardware above.
TASK & RESOURCE MANAGEMENT		
	BOINC	***Condor***
Resource allocation to Jobs	Server	Server
Task allocation	Server	Server
ROBUSTNESS		
	BOINC	***Condor***
Checkpointing	Yes when optioned by client software.	Yes when optioned by client software.
Scalability	Claimed: Millions of jobs per day. Observed: Millions of users per day.	Observed: Large pools of ostensibly dedicated processors up to 4800 reported in one site. 10000 concurrent jobs (Withers 2007).
Quality of service	Some features with dedicated machines.	Yes.
Churn	Yes. Readily.	Depending on application but generally yes.
Malfeasance	Multiple calculations, pattern fitting, extensible by the algorithm developer.	Secure communication, secure execution (sandboxing), multiple calculations.
LICENSING		
	BOINC	***Condor***
Licence Description	GNU Lesser General Public	Apache License 2.0

	License	
Source Availability	Open	Open
Governing Organisation	University of California	University of Wisconsin-Madison
USABILITY		
	BOINC	**Condor**
Hosting a new project	Server hardware dedicated to the task. High speed internet connection.	Server hardware dedicated to the task. LAN/WAN connection to processing nodes.
Joining an exiting project	Download software and pick a project.	Depending on usage. From downloading software and joining a project to dedicated machine.
Coding for a new project	Support for existing binaries and different levels of library integration.	Support for existing binaries and different levels of library integration. More work also depending on services required (i.e. PVM, MPI)
SUPPORT		
	BOINC	**Condor**
Commercial	None officially.	Yes.
Community	Yes. Mailing lists, Web sites, Wikis, Unofficial sites.	Yes. Mailing lists, Web sites, Wikis, Unofficial sites.
Books	None specific, but several scientific texts relating to it as a tool to use with some instructions.	None specific, but several scientific texts relating to it as a tool to use with some instructions.
PARTICIPATORY INCENTIVES		
	BOINC	**Condor**
Data Access	Per project basis.	Per project basis.
Financial through statistics	No.	No. But possible depending on grid middleware.
“Fame” through statistics	Yes.	No.

Table 42: WAN-DC qualitative results for both BOINC and Condor.

5.5 Discussion of Performance Results on BOINC and Condor

This section now follows on to compare and contrast the results of these systems now that the benchmark has been applied.

5.5.1 POV-Ray

Condor performed well with speedup close to the corresponding number of nodes in the cluster (when correcting for an outlier in 2 nodes). BOINC performed similarly to condor for the 1,2,and 4 machine cluster configuration (again when correcting for the 2 node outlier) however super linear speedup was observed for 8 and 16 node clusters. BOINC specifically supports processor hyper-threading support and multi-core support which explains this behaviour. In BOINC's case it will request and have 2 jobs allocated at a time per processor on the machine.

5.5.2 Transcode

At first glance, both Condor and BOINC appear to provide similar results for the Transcoding exercise across some cluster sizes. They both have similar speedup results across the range of cluster sizes. Elapsed time on BOINC is approximately twice as what was observed for Condor with the same job on the same hardware. This is to be expected given the differences between BOINC and Condor's job allocation where there is considerably more turns of communication for BOINC in order to have a job allocated.

5.5.3 No Work

On both systems little speedup was observed. In the case of Condor very little changed from 2 machines up to 16. On BOINC, speedup was observed, however this result is offset by the elapsed time which was far more than that observed on Condor.

5.5.4 Mandelbrot

Mandelbrot on Condor behaved like *No Work*. On BOINC, similarly to *Transcode*, the high overhead for work unit allocation resulted in much longer running times and, in this case, scope for speedup. Though in every case the elapsed time was less on condor than BOINC. However by 16 machines on BOINC there was significantly less difference between the two suggesting that the overhead on BOINC becomes less of an issue compared to Condor as its overhead increases along with the size of the cluster.

5.5.5 One Second

In both cases, each system showed their focus. Condor produced a set of results that would be expected for a system designed for more-or-less local area networks; not strictly so but certainly not the extremely transient network participants that BOINC was designed for. Results quickly converged and produced run times that were very consistent. BOINC, however, quickly degraded to a speedup “flat line” for such small jobs as they are simply not suitable for WAN calculation with BOINC.

5.5.6 Mean Work Unit

Here the difference between BOINC and Condor is stark and as expected. Condor demonstrates a strong link between the work load of a work unit and the time taken to complete this unit. This shows the general efficiency of job allocation and completion between the server and the client. A requirement for a system that has a focus of working as a LAN cluster. BOINC is optimised for WAN operation and as such, with the client focus of requesting jobs, shows a greater difference between the time taken to complete each work unit.

5.6 Chapter Summary

This chapter has presented the results of applying the WAN-DC benchmark to both the BOINC and Condor distributed computing systems. Both sets of experiments were performed on the same cluster hardware, network and data sets. This has produced a comparative set of results for two distributed systems that have not been compared in this way previously at time of writing.

These results form the basis of a comparison for a peer-to-peer distributed computing system that will be introduced in the next chapter and later subjected to the same benchmark under the same conditions.

Chapter Six

6.0 CompTorrent

This chapter describes the overall design and operation of CompTorrent. CompTorrent embodies the practical implementation of this thesis to demonstrate that a peer-to-peer network can produce comparable results to that of a client-server approach when used for general purpose distributed computing.

6.1 Introduction

CompTorrent allows a small group or an individual to host their own distributed computing project. This is achieved without needing to know much about distributed computing and, in many cases, without writing any new code. CompTorrent allows a group of nodes to share a dataset that needs to be computed. They share the original dataset, the computation load and the resulting computed dataset. This allows an originating node to upload an original data set only once and still share the entire dataset amongst many nodes. CompTorrent shows a partially decentralized peer-to-peer network being successfully used for distributed computing.

CompTorrent introduces several new techniques to distributed computing in order to solve some existing problems. Most importantly, and unfortunately also hardest to quantify, is the claim to lower the cost of entry to distributed computing from the perspective of those wanting to have something computed. Joining a computing project tends to be easy. However, starting one requires much more work. Many systems, such as those mentioned in Chapter 3, have a very simple means of joining the system – mainly the installation of some software and then the running of an application, which often presents as a screen saver for when the machine is otherwise unused. Others based on Java Web Start , again as described in Chapter 3, can be joined with the click of a URL. Any earlier difficulties perceived in joining a distributed computing project have very much been solved. However, the creation of a distributed computing project tends to be more difficult. BOINC, arguably one of the more open and easier systems to create with, still requires the dedication and configuration of server hardware to the task of managing a project. The Gnutella Processing Unit (See 3.3.2.1) does not natively allow your own projects to be created at all (GPU 2007). Many other systems exist that are dedicated to a particular task and would fall into the very difficult to create category. That is if you want to start a distributed computing project then first you must write a distributed computing system. While CompTorrent is not the first to introduce a generic distributed platform, as clearly BOINC and Condor have been shown to do this in this thesis, it is the first to utilize the tracker and “metadata file” concepts to attempt to satisfy the goal of making a system that is both easy to join and easy to create new projects.

An overview of this new system begins with the notion of a “seeder”, that is, the group or user who initiates the distributed processing task and has a full set of the original data. First they create a metadata file (using a software tool) which describes or contains the algorithm and describes the data set. This metadata file can be published on the World Wide Web (WWW) or another peer-to-peer (P2P) service,

for interested parties to download. This process is illustrated in figure 15 below.

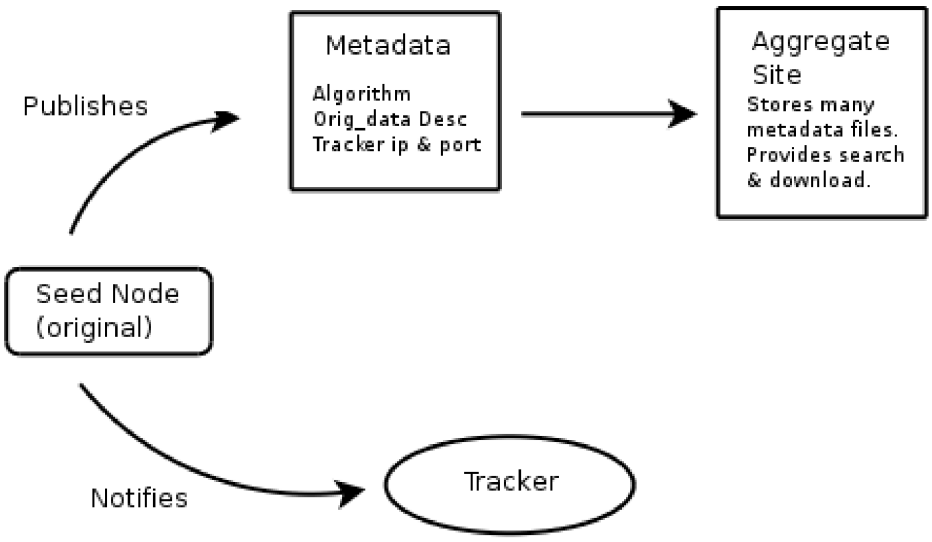


Figure 15: A new job is started by the creation and publication of a metadata file.

The distributed metadata file is what allows other users to join in the computing exercise. Once downloaded, another interested user uses the CompTorrent application to read this metadata file, extract the algorithm, begin computation and attempt to join the other computers working on the project. It does this by first contacting a "tracker" whose contact details are included in the metadata file. The tracker is a service hosted on the WWW which maintains information about which nodes are currently working on a problem and which parts of the problem are currently unsolved. The tracker suggests tasks for each node and helps coordinate the process. Its serves as a shared memory for the swarm and does so independently of it. This process is illustrated in figure 16. Here it is claimed that these techniques greatly simplify the task of starting a distributed computing project whilst also leaving it equally simple to join. It also allows separate computing jobs to be completely independent of one another so as to minimize any overhead in maintaining any other project other than your own. This approach contrasts with having a large group of nodes running multiple projects divided between them. Using file sharing as an example, BitTorrent's approach of a single swarm per file set easily outpaces Gnutella's approach of one large network with many file sets (Ritter 2006) (Bharambe 2005).

CompTorrent also introduces the notion of sharing the data set as well as the computation at the same time. Whilst a distributed system has always needed to share some of the data, namely the data being computed, here the incentive to join a project is to share in the computed data. Collaborative video encoding from a higher to lower bit rate can share the work and distribute the result at the same time. Using the output of one computing exercise among several research groups for the input of another is also a tangible incentive.

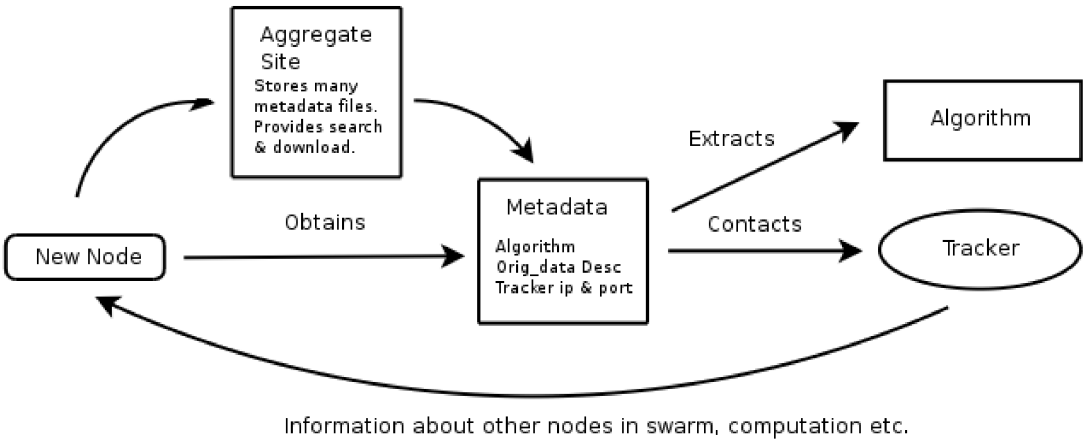


Figure 16: The major steps in joining a CompTorrent swarm

6.2 Technical Overview

Following the overview just given, this section proceeds to describe each major part of the CompTorrent system in more detail and show how each part interacts with the rest of the system. Security and trustworthiness of the overall system is also discussed.

6.2.1 Metadata File

The metadata file contains information about the location of the tracker, the algorithm to be used and a description of the original data set. It contains everything a new node needs to find the tracker to join the swarm, the algorithm used to compute a part of the result and the sizes, names and hashes of each piece of original data to be computed. This file is formatted in XML. An example is given below.

The first section of the file contains the version of the file, the connection details for the tracker, the name of the computation project, the size and hash of the original data set. The algorithm subset of the file contains the execution details of the algorithm and the algorithm itself. There are two broad options for the algorithm in the metadata file. The swarm can rely on the algorithm application being available on the participating machines (as is shown in the example given) or the application binary can be directly embedded into the metadata file in a base64 encoding. Either way, this approach allows the seeder to distribute the algorithm as flexibly as possible. Java bytecode is easily included or a more complicated script can be used to broadly cater for a variety of situations and platforms. Once the algorithm has been extracted or obtained, the execution field stipulates how the algorithm is executed. It is assumed that there will always be an input file that contains data that will be acceptable to the algorithm. A resulting computed data set will be produced and saved in a computed directory.

```
<?xml version="1.0" ?>
<comptorrent>
<version>0.1</version>
<tracker_url>144.6.40.251</tracker_url>
<tracker_port>60000</tracker_port>
<name>cruelcruellove</name>
<size>93130756</size>
<md5>8EE44CB5C9A5AFCACD6C0AF363C1C5A1</md5>
<algorithm>
<execution>algo.sh</execution>
<script>
#!/bin/sh
transcode -i $1 -o $2 -y xvid
</script>
</algorithm>
<orig_data>
<file><name>chunk-
001.mpg</name><size>1035738</size><md5>055172279073E1DC42C847BC794816A5</md5></file>
...
<file><name>chunk-
100.mpg</name><size>703680</size><md5>BCEB81C97C89B6C0D61CFC8F8F1384DE</md5></file>
</orig_data>
</comptorrent>
```

The remaining section of the file describes the original data set. It does not contain the original data, only its representation in terms of name, size and hash. Nodes ask each other for original data as necessary and share the bandwidth load of the distribution task. The size of each data chunk in the set is dependent on the nature of the job and left to the judgement of the seeder at the time the metadata file is created. A typical data chunk size may be in the range of 256kb to 1Mb depending on the intensity of the computation task and quality of the network connection between typical nodes. Just like the algorithm, this data can be sent in plain text or Base64 encoded depending on the nature of the data to be processed. Anything that is acceptable to XML can be left in original form whilst binary data can be encoded or compressed and encoded. Along with each chunk of data, the size of the data and a hash of the data for checksumming is included. Support for very large data sets is available now via two means. One option is by having multiple CompTorrent metadata files which swarms can join to process parts of the dataset independently. A second is that the embedded or downloaded algorithm manages is own data handling.

6.2.2 Tracker

The tracker is a WWW service that provides a simple shared memory for a swarm or number of swarms. From the tracker a node can get a list of other connected nodes in the swarm, get a suggestion for the next data chunk to process and report data chunks finished. It is a simple service that is basically a web-based front end for an SQL database to allow nodes to gain and provide information quickly. A tracker is kept simple and provides no significant processing services so that a swarm need not completely rely on it for its work. As such a node does not necessarily keep an open connection to the tracker at all times. It connects and makes requests as needed. There is no reason why the tracker could not be ported to other mediums beyond HTTP. This is the subject of future work and is discussed at the end of this thesis.

In a traditional client-server distributed computing system, the services provided by a Tracker would be offered by a dedicated server. In a hybrid system such as this one, any tracker, or multiple trackers, can be used to provide these kinds of services. For general purpose distributed computing, the usage of the tracker paradigm by CompTorrent is novel.

The current implementation of the tracker includes tools for gathering and disseminating much operational data. Data are made available on which nodes have

which data pieces and at what time files were received or calculations made and at what times were connections between nodes made. All of these data are displayed in a web based application. Real time graphs of network topology are available as are visual indicators of original and computed data per node.

6.2.3 Node

Nodes speak a simple protocol that is represented in XML and communicate via sockets. This protocol is original and is not compatible with any other peer-to-peer protocol. XML was chosen due to ease of ongoing modification to the protocol when compared with a binary message structure style approach. Nodes make connections with each other after asking a tracker for nodes that are already in the swarm and how many existing connections to other nodes they already have. Each node will make as many outgoing connections to other nodes and receive incoming connections as it can based on user configuration. This overlay network is maintained for the life of the swarm as new nodes join and existing nodes leave. Presently, connection candidates are suggested by the tracker based on the simple heuristic of choosing the least connected node from a pool of nodes that do not have connections which involve the new potential node. A routing scheme is then overlaid the underlying TCP/IP network.

The communication protocol is simple and largely consists of messages to manage connections and exchange data chunks. Connection requests include information about what original and computed data a prospective partner node has and details about which other nodes it is already connected to. A node will then accept or refuse a connection with a reply and pass back similar information to take advantage of this brief connection. Connected nodes pass file request and file reply messages back and forth as they work towards completing their datasets.

A node computes a part of the overall job and reports to the tracker that it has finished. Nodes make requests to each other to ask for parts of the original and the computed data sets. Once a node obtains a new chunk of data, it reports this to the tracker so it can service requests for that chunk as well to help share the load.

Each node is equivalent to every other node in the network and has no different functionality whether it be an originating seed or a new node joining a large existing swarm. Nodes are arbitrary volunteers on a network and do not need to have similar architectures. Every node computes and shares data with every other node that it is connected to. There are no “special” nodes with greater importance to the swarm or different responsibilities. It is the equal aim of each node to assemble, and maintain, a complete set of the computed datasets and optionally the original data set in an attempt to provide as much redundancy as possible to the swarm as a whole. Computed chunks that are lost can be recalculated, allowing the swarm to heal itself should a node leave without sharing its computed data. Original data are replicated quickly amongst nodes in a rarest first fashion. To further illustrate this process, figure 17 shows a seed node with a full original data set and half of the computed set. Node 1 has obtained half of the original data and applied the algorithm producing some computed data. The tracker helps direct each node to form an overlay network and suggests chunks for computation and sources for data. As the computation is finished, the new node would work with the seed and node 1 for copies of the original and/or computed data.

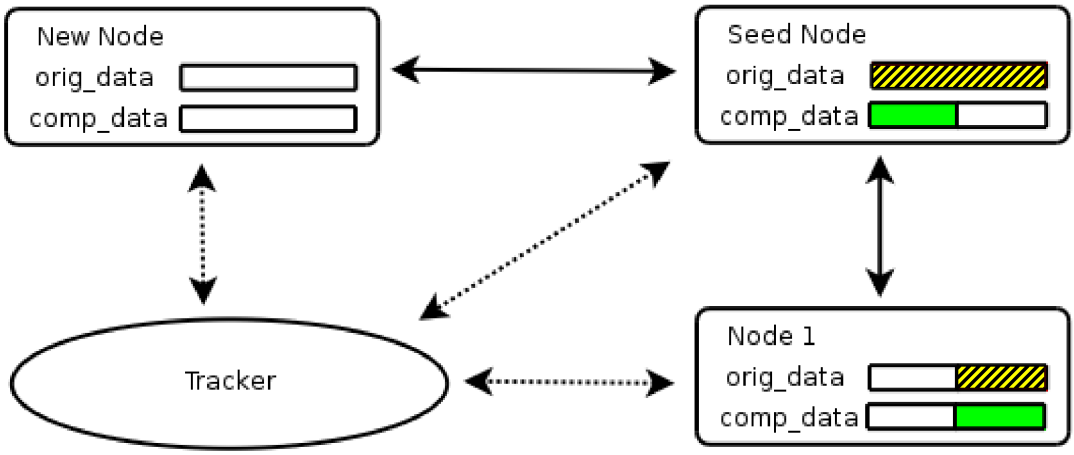


Figure 17: Three nodes is a simple network interacting with each other and the tracker.

6.2.4 Security

Whilst not actually a discrete part of the system like the tracker or the node, the security implications of the system need to be considered in order to gauge its usefulness. From the description of the system already given, it is clear that the behaviour of nodes can have a dramatic impact on the reliability of the system. The major features currently implemented which addressing these issues will now be discussed.

To begin with there is an implicit trust in the seeder or the group who has constructed the metadata file and a new user wishing to participate. This is especially the case when a custom or unknown algorithm is the agent of computation. This is where the ability to include a script to use an existing application might be more suitable. It is envisaged that in time, users would be able to gain credibility, based on the quality and trustworthiness of their offerings, that would be manifested in a community that has grown around the distribution of the metadata files itself. This is certainly what has occurred with BitTorrent where there are many search or aggregation sites which serve as databases for existing BitTorrent swarms. Users are commonly allowed to make comments on each file available.

Once there is some measure of trust in the seeder, the integrity of the metadata file itself, whilst not currently implemented, could be managed with a digital signature scheme using existing tools. Original data integrity is already managed with hashes and this would obviously be further strengthened if the metadata file, containing the original set of hashes, was digitally signed by the author.

The computed datasets are easy targets for malfeasance. The hashes given in the metadata file protect the original data, but that does little to suggest the integrity of their computed equivalent. In CompTorrent, a seed stipulates how many times each data chunk is to be recomputed, by a separate random node, before it is considered trustworthy. This clearly has a profound effect on the time needed to compute an entire set. However in an uncontrolled environment it is one of the few tangible ways to get an idea of how much trust can be placed on a result. Other distributed systems commonly use various credit and cheating techniques to manage and rate node contributions; this is something that may be examined in the future for

inclusion beyond the re-computation which is currently implemented.

6.3 Using CompTorrent

This section demonstrates the usage of CompTorrent and forms a part of the method employed in Chapter 8 for obtaining measurements. This is similar to Appendix C where a method for reproducing the results described in Chapter 5 is given for Condor and BOINC.

6.3.1 A Suitable Algorithm (and data!)

There are three stipulations for an algorithm to be computed with CompTorrent as described in this thesis:

1. The algorithm being computed must be independently parallel.
2. The computation time should be larger relative to the network time required to distribute the data for the exercise to be worthwhile.
3. The interface with the algorithm, as an executable, should take an input file, execute, and then produce an output file in a synchronous fashion.

Obviously a data set would be a necessary requirement. It should be divided into suitable portions so as to satisfy item number 2 above and be suitable to be passed to the algorithm.

Now an algorithm that meets these requirements has been written or obtained, a tracker needs to be found to host the swarm.

6.3.2 Locating a Tracker

A tracker can be either controlled by the owner of the project or not. Providing the tracker is willing to host the project. All that is required is to know where the project is to be ultimately hosted in order to move on to create the metadata file.

6.3.3 Creating the Metadata File

A metadata file needs to be created as per 6.2.1. There is a tool that has been written to make this a less arduous task. This tool is imaginatively called makecomptorrent and has the following signature:

makecomptorrent ***file_name directory_name tracker port algorithm execution max_chunk_size***

filename: the file name of algorithm that does the work)

directory_name: The directory containing the original data files that are to be computed (and nothing else)

tracker: The url that the tracker is hosted on

port: The port that the tracker is on the url you just gave

algorithm: The executable file that actually does the work

execution: How CompTorrent is supposed to run the algorithm?

max_chunk_size: The maximum data size for original or computed data set pieces (in bytes)

For example:

```
./makecomptorrent onesecond jobs 144.6.40.251 60000 onesecond.exe onesecond 256000 > onesecond.comptorrent
```

This will create a project called *onesecond* using the files in the `jobs/` directory using a tracker hosted at 144.6.40.251:60000. The executable `onesecond.exe` will be used at the algorithm and the maximum file size of the data calculated. The metadata file is output to `stdout` so in this example it will be redirected to the file *onesecond.comptorrent*.

6.3.4 Planting the Seed

There are two steps to this. Firstly, a seed node on a machine somewhere needs to be started that will generally stay online long enough for the complete dataset to be uploaded to the swarm. The seed is started thus:

```
comptorrent metadata_file network_interface tracker_url (optional)  seed compute
```

metadata_file: The comptorrent metadata file for the swarm you want to start (or join).

network_interface: Which network connection to use.

tracker_url: This is optional, however it is possible to use a different tracker to the one stipulated in the metadata file.

Seed: 0 or 1. Am I the seed (and running for the first time)? The only real difference here is that it will ask the tracker to clear the cache of data for the job.

Compute: 0 or 1. Will this node be contributing computing resources or just hosting data files. This is useful for seeds which do not want to do any of the actual computing work.

For example:

```
./comptorrent comptorrents/mandelbrot.comptorrent eth0 1 0
```

This will start the application using the *mandelbrot.comptorrent* metadata file, listen on the network adapter called *eth0* as the seed and not do any computation.

Now that the seed is running, connected to the tracker and listening for new connecting nodes, nodes need to be encouraged to connect. The first and most obvious step of this is that the metadata file will need to be shared. In a controlled environment it can be shared on a local drive or server; in a WAN environment a web site or via email may be more appropriate. Nodes who wish to participate download the file and start the application in the same way, though with the last two options reversed, as they are not the seed and you would like to hope that they will compute as well.

6.3.5 Seeing the Results

At present the tracker is open for all users to see the computing progress. Some differentiation between the project owners and participants later on would be an obvious requirement for production. A web based interface was implemented that describes the overall swarm and its progress.

6.3.5.1 The home page

This page lists all of the jobs hosted on the tracker and provides a menu to each major function.

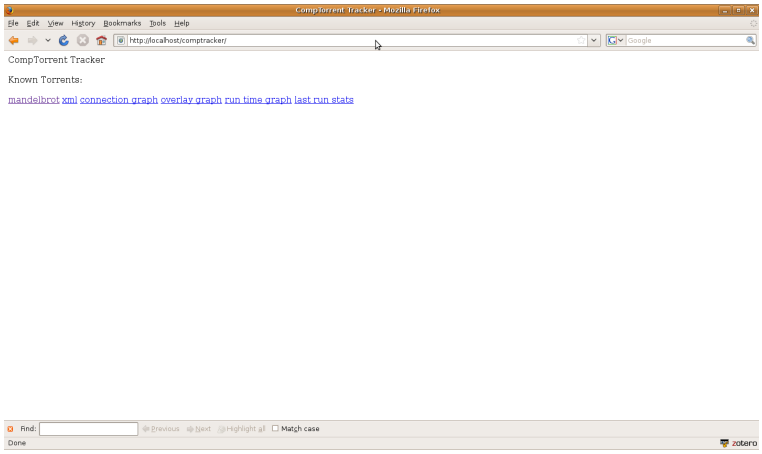


Figure 18: The tracker WWW interface home page.

6.3.5.2 Node & Work Unit List

The node and work unit list (accessible by clicking on the name of the job in the diagram above) gives an overview of the job's progress. The nodes in the swarm are described as are the statuses of each work unit. Work units can be observed as uncomputed (red), allocated and in-progress (yellow) and completed (green). Figure 19 shows a completely computed project.

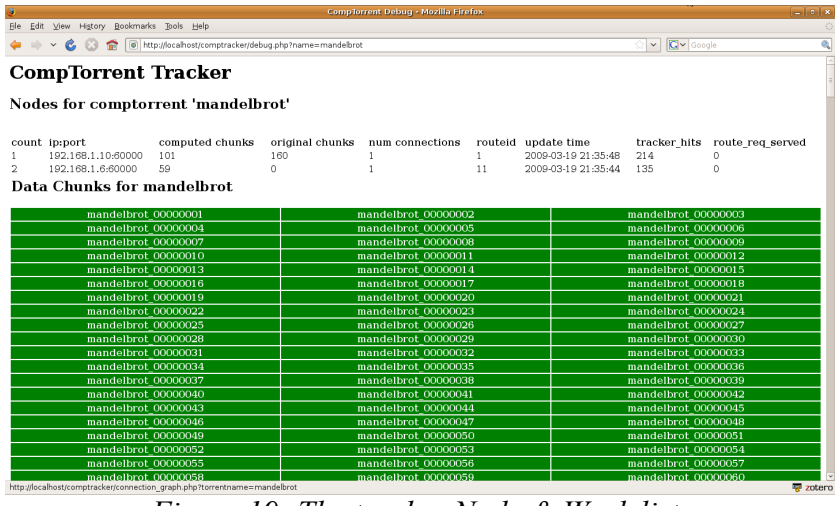


Figure 19: The tracker Node & Work list.

6.3.5.3 Overlay Graph

The overlay graph shows a representation of the overlay network tree for the swarm. Each node is allocated a network overlay key which allows it to route messages to any other node in the overlay network without keys needing to be allocated by any central authority. This is discussed in further detail in section 7.1.2.3.

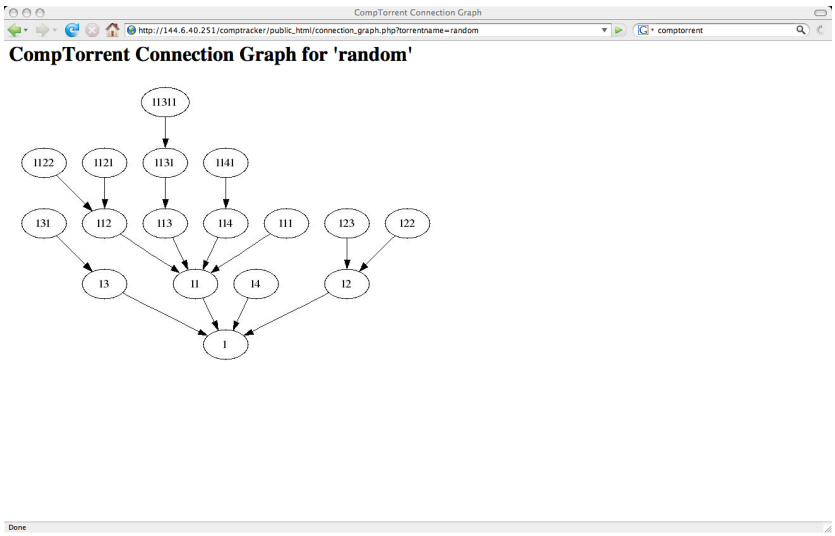


Figure 20: The overlay graph

6.3.5.4 Connection Graph

The connection graph shows the IP connections between each node in the swarm. It shows the actual network socket connections between each node that are made.

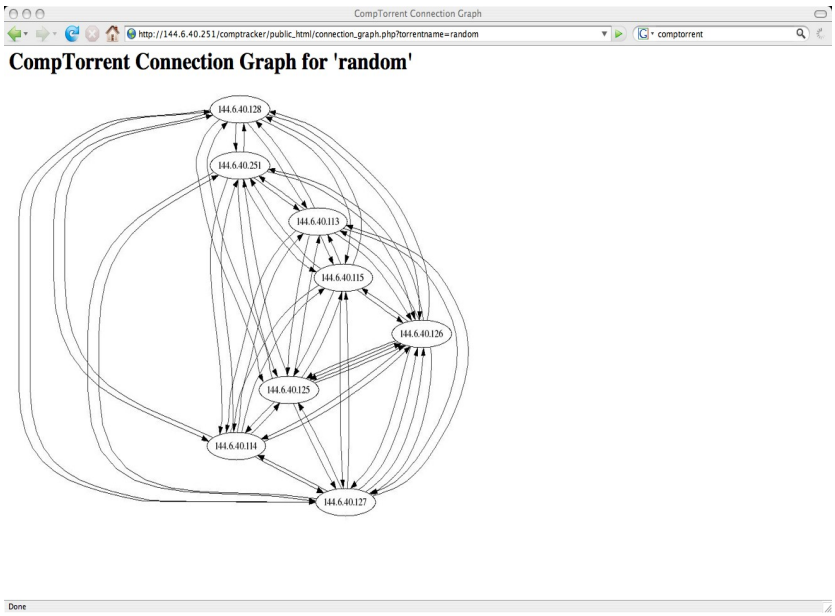


Figure 21: The IP connection graph

6.3.5.5 Last Run Times

The Last Run Times page primarily shows a list of each work unit, when it was completed, when nodes joined the swarm and so on.

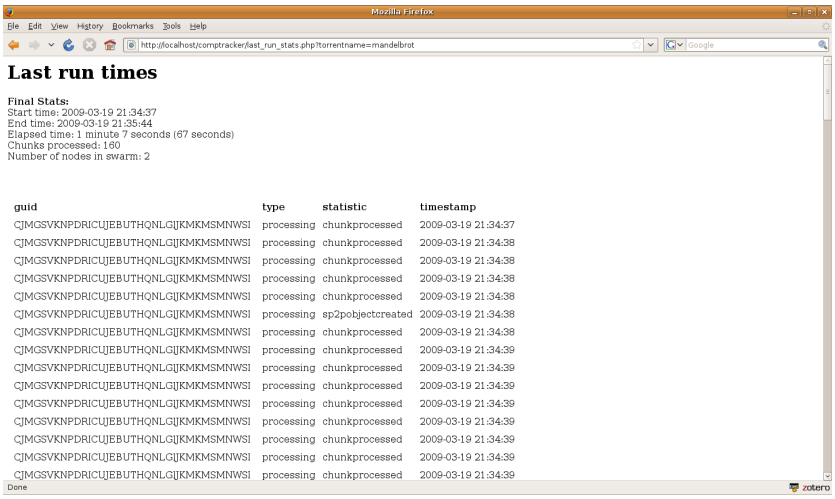


Figure 22: The tracker page of last run times and other statistical information.

6.4 Chapter Summary

This chapter has presented a high-level description of a new distributed computing system that is relatively generic and easy to use for both joining and creating a distributed computing project. Techniques have been applied to distributed processing that have not been applied before, namely the metadata file and tracker paradigms, that have produced favourable results. This has allowed for CompTorrent to use many existing compiled programs without modification as an algorithm for a computing swarm to use.

Chapter Seven follows with a detailed description of CompTorrent.

Chapter Seven

7.0 CompTorrent Implementation

This chapter will discuss the individual components that comprise the overall CompTorrent system in detail.

7.1 Major Components

As discussed in the previous chapter, CompTorrent consists of two major software components, the tracker and the CompTorrent peer-to-peer application itself. This section will examine each in detail.

7.1.1 Tracker

This section builds on the description in section 6.2.2 to provide a detailed technical description as well as a discussion of design considerations. The Tracker is a WWW service that provides a discovery service for nodes as well as a repository for what work has been done. It is a collection of web scripts written in the PHP language and backed by a MySQL database.

7.1.1.1 Communication Protocol with Nodes

Nodes communicate with the tracker via a HTTP interface. Requests are made to individual files on the tracker server. Each file has a specific purpose, set of request variables (in our implementation using a GET[14]) and reply. Each is discussed in turn in the table below to illustrate the overall process.

Each of the functions described here has a related database table, or set of tables, which is described in the tracker database schema in Appendix A.

`chunk_report.php`	**Purpose:** To advise the tracker that a work unit has been completed. **Parameters:** uuid – The unique id of the node. Used as an identifying key. name – The name of the work unit. hash – The hash generated from the original data. comptorrentname – The job that the work unit belongs to. comp_state – The current status of the work unit {0:uncompleted, 1:allocated/in-progress, 2:completed}. Generally a node will only set it completed as the tracker will manage the uncompleted and allocated states.

14 A GET is a part of the HTTP specification which allows data to be passed to a web server. See the HTTP/1.1 specification for more information (RFC 2616).

	torrenthash – The md5 hash of the computed result. **Return Values:** boolean – success or failure of request. **Primarily Related Database Table(s):** sp2p_datachunks
register_node.php	**Purpose:** To advise the tracker that a new node on the network for a particular comptorrent. Sent by the joining node as soon as it has been allocated a route key by another node. **Parameters:** ip – The node's IPv4 address. port – The node's Ipv4 port for listening for new join requests. uuid – The unique id of the node. Used as an identifying key. name – The name of the "comptorrent" or computing job. So a tracker can host multiple jobs. routeid - The key in the route table that has been assigned to this node. **Return Values:** boolean – Success or failure of request. **Primarily Related Database Table(s):** sp2p_node
node_report.php	**Purpose:** To advise the tracker of a node's ongoing status. **Parameters:** ip – The node's IPv4 address. port – The node's Ipv4 port for listening for new join requests. num_connections – The number of total connections to other nodes in the network. uuid – the unique id of the node. Used as an identifying key. comptorrentname – The name of the "comptorrent" or computing job. So a tracker can host multiple jobs. num_computed_chunks – The amount of computed work units that this node has in it's possession. num_original_chunks = The amount of original data subsets that this node has in it's possession. routeid - The key in the route table that has been assigned to this node. **Return Values:** boolean – success or failure of request. **Primarily Related Database Table(s):** sp2p_node
clear_records.php	**Purpose:** To clear the records for a tracker. Used primarily as a tool in

	testing and running multiple computations with the same name repeatedly. Not particularly suitable for deployment in a production environment. **Parameters:** None! **Return Values:** None! **Primarily Related Database Table(s):** All tables.
get_comp_hash.php	**Purpose:** To ask the tracker for the computed hash of a particular work unit. This can aid the node in assessing the reliability of a work unit being received from another node. **Parameters:** nodeuuid – The unique id of the node. Used as an identifying key. chunkname – The name of the work unit. **Return Values:** string – MD5 (or other) hash of the accepted computed hash for this work unit as determined by the tracker. **Primarily Related Database Table(s):** sp2p_datachunks, sp2p_computing
random_chunk.php	**Purpose:** Request a work unit to process. Random may be a misnomer as the tracker can employ a variety of algorithms besides randomness to make a suitable suggestion. Algorithms include in order processing (start at the first and work through to the end), random, least computed, and least confirmed. **Parameters:** nodeuuid – the unique id of the node. Used as an identifying key. **Return Values:** string **Primarily Related Database Table(s):** sp2p_datachunks
stats.php	**Purpose:** Report a generic operational statistic. This is used to gather data on the nature of the computation, when nodes join and leave, when the are computing a work unit and so on. **Parameters:** nodeuuid – the unique id of the node. Used as an identifying key. type – A free-form string of up to 12 characters to describe the type of statistic e.g. processing, init, etc. statistic – A free-form string of up to 64 characters to contain

	statistic data. **Return Values:** boolean – success or failure of request. **Primarily Related Database Table(s):** sp2p_stats
file_report.php	**Purpose:** Report having an original data subset. **Parameters:** torrenthash – The identifying hash of the comptorrent. nodeuuid – The unique id of the node. Used as an identifying key. resulthash – The MD5 hash computed from the original data subset. filename – The name of the work unit. **Return Values:** boolean – success or failure of request. **Primarily Related Database Table(s):** sp2p_files
finish_report.php	**Purpose:** Report a computed work unit. **Parameters:** orighash – The identifying hash of the original data subset. torrenthash – The identifying hash of the comptorrent. uuid – The unique id of the node. Used as an identifying key. resulthash – The MD5 hash computed from the work unit. chunk_name – The name of the work unit. **Return Values:** boolean – success or failure of request. **Primarily Related Database Table(s):** sp2p_datachunks, sp2p_computing, sp2p_files
set_chunks_done.php	**Purpose:** Report a number of computed work units in one message. Communicating with the tracker can be expensive. This attempts to minimise the cost. **Parameters:** comphash – Multiple md5 computed work unit hashes as comma separated values. torrenthash – The identifying hash of the comptorrent. nodeuuid – the unique id of the node. Used as an identifying key. **Return Values:**

	boolean – success or failure of request. **Primarily Related Database Table(s):** sp2p_datachunks, sp2p_computing, sp2p_files
set_orig_data.php	**Purpose:** Similarly to set_chunks_done.php this reports multiple original data set pieces. **Parameters:** origdata – Multiple md5 computed original data set hashes as comma separated values. torrenthash – The identifying hash of the comptorrent. nodeuuid – the unique id of the node. Used as an identifying key. **Return Values:** boolean – success or failure of request. **Primarily Related Database Table(s):** sp2p_files
report_connection.php	**Purpose:** Advise the tracker of a connection between two nodes in the overlay network. **Parameters:** nodeuuid – the unique id of the node. Used as an identifying key. server - The unique id of the node acting at the “server” end of the connection. client - The unique id of the node acting at the “client” end of the connection. torrentname – The name of the “comptorrent” or computing job. So a tracker can host multiple jobs. **Return Values:** boolean – success or failure of request. **Primarily Related Database Table(s):** sp2p_connections
request_node.php	**Purpose:** Request a candidate node to make a connection with. The tracker can employ a variety of algorithms to make a suitable suggestion. **Parameters:** uuid = the unique id of the node. Used as an identifying key. ip = the IPv4 address of the node making the request. **Return Values:** string – IPv4 address and port of suggested node. **Primarily Related Database Table(s):** sp2p_node, sp2p_connections

get_route_to_file.php	**Purpose:** Ask the tracker for the overlay key of the node most likely to have the requested file. **Parameters:** nodeuuid – The unique id of the node. Used as an identifying key. chunkhash – The computed MD5 hash of the requested file. **Return Values:** string – route id of a node with the file. **Primarily Related Database Table(s):** sp2p_node, sp2p_files, sp2p_datachunks
suggest_orig_chunks.php	**Purpose:** Ask the tracker for some suggested original data set files. These can be allocated randomly or on the least shared files. **Parameters:** lastchunk – The last original data set file suggestion that the node retrieved from the tracker. This can hint the tracker based on what files the node reports to already have. It can suggest files around there – or not around there. Depends on the algorithm being used. **Return Values:** string – work unit suggestion **Primarily Related Database Table(s):** sp2p_datachunks
report_ipconnection.php	**Purpose:** Report to the tracker a connection between nodes that has been made where they already have overlay keys. These redundant connections are made to improve network robustness and performance. At the tracker level, these notifications are a courtesy (and allowance for research so the actual network can be drawn) as nodes will "short-cut route" messages at each node without communication with the tracker. This is discussed further in 7.1.2.3. **Parameters:** server – The IPv4 address of the node at the "server" end of the connection. client – The IPv4 address of the node at the "client" end of the connection. nodeuuid – the unique id of the node. Used as an identifying key. **Return Values:** boolean – success or failure of request. **Primarily Related Database Table(s):** sp2p_ipconnections
ipconnection_exists.php	**Purpose:**

	Ask the tracker whether two nodes have a IP connection. Helps nodes make decisions about whether to accept incoming requests. **Parameters:** server – The IPv4 address of the node at the "server" end of the connection. Can be thought as address 1 as it will test for connections made in either client-server or server-client configurations. client – The IPv4 address of the node at the "client" end of the connection. Can be thought as address 2 as it will test for connections made in either client-server or server-client configurations. nodeuuid – the unique id of the node. Used as an identifying key. **Return Values:** boolean – connected or not connected. **Primarily Related Database Table(s):** sp2p_ipconnections

Table 43: The major web based components of the CompTorrent Tracker.

7.1.1.2 Discussion of Tracker Design

The approach taken here, using a web application along with a MySQL database, affords all of the robustness that a modern web application enjoys. It is possible therefore, with little extra application work, to take advantage of load balancing between servers and replication of databases.

The Tracker itself only aids the computation through suggestions of work units when nodes request them. These suggestions aid swift computation however nodes can request and compute any available work units they wish. This can improve performance as the tracker will allocate work chunks based on what files a node already has to minimise processing starvation – without the node knowing what has or has not been computed. This approach minimises the amount that the nodes need to know about the overall network and removes the associated bandwidth between nodes over the network that would otherwise be required for this functionality. The Tracker will also direct the job through the nature of the computation and nature of the network (in order processing (start at the first and work through to the end), random, least computed, and least confirmed). A tracker will also periodically reassign work that appears to have taken longer than the average.

The tracker collects information reported from nodes. All information reported is specifically used for aiding the computation task such as nodes making connections to other nodes, their overlay keys and which data have been processed. Further derived from this information, are statistics and statuses which are used for informational and testing purposes.

A tracker also enforces the swarm paradigm and naturally limits the size of the peer-to-peer network. Nodes participating in a particular job rather than nodes participating in a large network where jobs are then chosen. This contrasts with an approach of a monolithic system such as a classical Grid or that taken by GPU (3.3.2.1).

7.1.2 CompTorrent Application

The reference implementation is written in C++ and utilizes the commonc++, crypto++ and tinyxml libraries. The application and all of the libraries used are compilable with the GNU C++ compiler ensuring that it is relatively portable between all major platforms.

Obtaining and compiling the code is detailed in Appendix A.

7.1.2.1 Major Objects

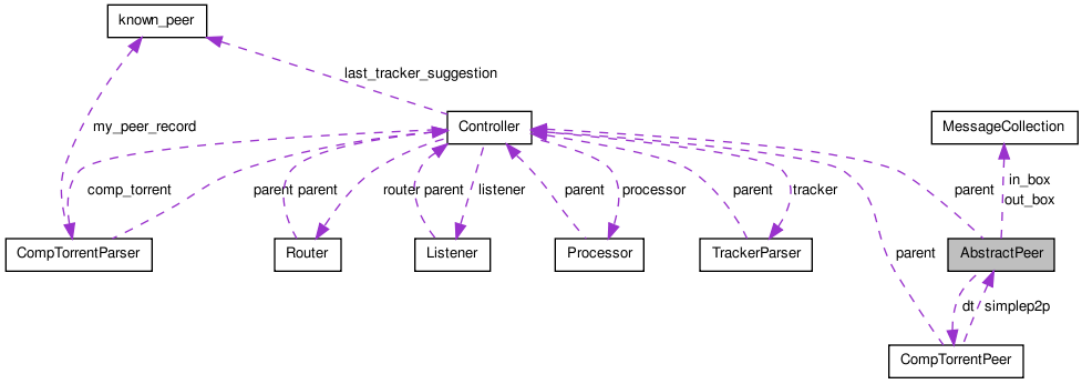


Figure 23: Collaboration graph for the sample implementation of CompTorrent

There are 8 major parts to the CompTorrent application. Most are objects that contain threads so that they can operate independently of other parts of the application. Some objects are in a hierarchical relationship and communicate with each other via inheritance or aggregation. Others communicate via a system of message queues so that they can service requests without needing to wait any longer for processing than absolutely necessary. Communication between the tracker and other nodes on the network is always “best effort” in this implementation. Resends of data and reallocations are dealt with via timeouts and subsequent reallocation of unsent work.

UML class diagrams are listed in Appendix B.

7.1.2.1.1 Main Loop

The main loop of the application is primarily concerned with bootstrapping onto the network and starting each of the main objects. It then periodically attempts to make outgoing connections.

```
if (Controller.parse_torrent(comptorrent_file)) {
  // decode the comptorrent file, extract the algorithm, create the
  //    working dir, etc
  Controller.create_working();
  Controller.start_distributed(my_ip, my_port);
  // get in touch with the tracker (hopefully) and see what's going
  // on with the wider network
  Controller.bootstrap_from_tracker();
  // start the thread which will wait for and manage incoming
  // connections
  Controller.start_listener(my_ip, my_port);
  for (;;) {
    Controller.tracker->report_stats_to_tracker();
    Controller.attempt_outgoing_connection(my_ip, my_port);
  }
}
```

7.1.2.1.2 Controller

The Controller class oversees the runtime operation of the application without needing to know too much about the actual protocol being run. It has the notion of connections, messages and files that need to be managed. It is the parent of all CompTorrentPeer objects created (CompTorrentPeer is described in the next section).

Controller contains several of the other objects described in this section which are created to make or receive connections on the network as well as parse the metadata file.

```
void start_distributed(string host_ip_, string host_port_);
void start_listener(string host_ip_, string host_port_);

void make_connection(string host_ip_, string host_port_);
void attempt_outgoing_connection(string host_ip_, string host_port_);
void attempt_incoming_connection(string host_ip_, string host_port_);

bool parse_torrent(string file_path);
bool create_working();
unsigned long bootstrap_from_tracker();
void clear_tracker_data();
```

There are also methods for general connection management which Controller maintains an overseer role in.

```
void add_connected_peer(string ip, string port, void* CompTorrentPeer);
bool connected_peer_exists(string ip, string port);
void out_broadcast(string msg);
bool remove_connected_peer(string ip, string port);

void add_peer_under_consideration(string ip, string port);
bool peer_under_consideration(string ip, string port);
bool remove_considered_peer(string ip, string port);

void set_routing_id(string routing_id_);
string get_routing_id();
string get_next_routing_id();

void set_ip(string ip_) { ip = ip_; }
void set_port(string port_) { port = port_; }
string get_ip() { return ip; }
string get_port() { return port; }

int get_num_connections();
int get_num_incoming_connections();
int get_num_outgoing_connections();

void increment_num_incoming_connections();
void increment_num_outgoing_connections();
void decrement_num_outgoing_connections();
void decrement_num_incoming_connections();
```

References to major objects are kept here.

```
CompTorrentParser* comp_torrent;
TrackerParser* tracker;
// NUM_PROCESSORS is how many units to work on concurrently
Processor* processor[NUM_PROCESSORS];
Router* router;
Listener* listener;
```

As with connection management, Controller oversees the node's collection of work units and files.

```
bool have_comp_chunk(string chunkname);
void add_comp_chunk(string filename, string resulthash);
```

```
void get_known_comp_chunks_xml(ostringstream& xml);
int get_num_comp_chunks();

bool remove_required_comp_chunk(string chunk_name);
void add_required_comp_chunk(string filename);
bool get_next_required_comp_chunk(file_chunk& f);
int num_required_comp_chunks();

bool have_file(string chunkname, string filetype);
void request_file(string chunkname, string filetype);

bool have_orig_chunk(string chunkname);
void add_orig_chunk(string filename, string resulthash);
void get_known_orig_chunks_xml(ostringstream& xml);
int get_num_orig_chunks();

bool remove_required_orig_chunk(string chunk_name);
void add_required_orig_chunk(string filename);
void add_required_orig_chunk_front(string filename);
bool get_next_required_orig_chunk(file_chunk& f);
int num_required_orig_chunks();
```

7.1.2.1.3 CompTorrentPeer

The CompTorrentPeer object acts on the messages received from other nodes (as described in 7.1.2.2). These messages are received and sent via message queues *in_box* and *out_box*.

It has methods which manage the sending of messages:

```
void send_welcome_message();
void send_noop();
```

As well as methods to parse received messages:

```
bool process_noop(string intmp);
bool process_file_update(string intmp);
bool process_welcome_message(string intmp);
bool process_connect_message(string intmp);
bool process_file_reply(string intmp);
bool process_file_request(string intmp);
bool process_overlay_message(string intmp);
bool process_overlay_update_message(string intmp);
```

Each of these methods directly correspond to the messages the process which are explained in 7.1.2.2.

Its run method embodies the logic for how these messages are marshalled.

```
void CompTorrentPeer::run() {

        while (!die) {

                bool message_sent = false;

                // if I am the server end of this connection and I'm unconnected,
                // send a welcome message

                if (state == p_unconnected) {
                        send_welcome_message();
                        state = p_overlay_wait;
                }

                string intmp(""); // the message popped of the incoming messages queue
                while (AbstractPeer->in_box.pop(intmp)) {
                        process_noop(intmp);
                        process_connect_message(intmp);
                        process_welcome_message(intmp);
```

```
                process_overlay_message(intmp);
                process_file_update(intmp);
                process_file_reply(intmp);
                process_file_request(intmp);
        }
}

...
```

It also manages link failure detection and kills itself if a connection becomes unresponsive

```
...
}
```

7.1.2.1.4 TrackerParser

The tracker object manages communication between a node and the Tracker. It also manages collections of some of the inter-node communication that the node has been exposed to.

A collection of methods to interface with the tracker interfaces as described in 7.1.2.1.

```
bool report_ipconnection(string client, string server);
bool ipconnection_exists(string client, string server);
bool get_route_to_file(const string chunk_hash, string&
routeid);
...
```

As well as methods to maintain some collections of information gleaned from communication with other nodes as per messages 7.1.2.2.

```
void add_known_peer(known_peer kp);
bool known_peer_exists(known_peer kp);
```

It maintains collections of the known_peers and data units.

```
vector<known_peer> known_peers;
vector<orig_data_chunk> data_chunks;
```

7.1.2.1.5 CompTorrentParser

The comptorrentparser object contains the data primarily gleaned from the comptorrent metadata file. It also manages collections of some of the inter-node communication that the node as been exposed to.

It contains methods to load and initialise the object from the metadata file:

```
bool load_xml(string);
bool create_working();
```

As well as methods to access parsed data:

```
string get_version();
string get_tracker_url();
string get_tracker_port();
...
```

It also manages the known connections between the node and other nodes connected as well as the collections of nodes that it have been previously known to exist on the network.

```
bool known_peer_exists(known_peer kp);
void add_known_peer(known_peer kp);
void remove_known_peer(known_peer kp);
...
unsigned long get_num_known_peers();
```

```
unsigned long get_num_connected_peers();
...
vector<known_peer> known_peers;
```

Methods exist to manage the data being processed. Collections of the original and computed data sets are kept.

```
vector<orig_data_chunk> data_chunk_list;
...
bool comp_data_exists(string filename);
bool orig_data_exists(string filename);
...
void   update_comp_chunk(string chunk_name, computation_state state, bool
file_exists);
void   update_orig_chunk(string chunk_name, computation_state state, bool
file_exists);
...
filemap known_files;
```

7.1.2.1.6 Listener

The listener manages incoming connections to a node. One is created for each "server-side" connection that a node accepts. Once the IP connection is made it is up to the created CompTorrentPeer objects to decide where a logical connection occurs. Here the only checking it to ensure that the node doesn't already have a connection to the requesting node. This can occur when two nodes both decide individually to make a connection to each other concurrently. There is a critical section in the listener that ensures that IP connections are handled one at a time to avoid superfluous connections.

```
void Listener::run() {

for (;;) {
  if (server.isPendingConnection()) {
  AbstractPeer* p2p = new AbstractPeer(server, parent, SERVER);

  if (!parent->connected_peer_exists(kp.ip, kp.port)) {
    parent->add_connected_peer(kp.ip, kp.port, (void*)p2p);
    p2p->set_my_ip(host_ip, host_port);
    p2p->detach();
  } else {
  server.reject();
  }
}
```

7.1.2.1.7 Router

The router exists in the Controller object which manages data chunks received by the node. When messages containing work unit data are received, they are placed in a queue for the router to manage. Providing it has a route the router will process original data chunks as they arrive (storing them on the node's disk and presenting them to the Processor object) and if the node is waiting on a request for a work unit computed by another node, process them as well by saving them to the disk.

```
for (;;) {
        if (parent->get_routing_id() != NO_ROUTE) {

        // grab the next required original file off the stack
        get_next_orig_chunk();
        if (parent->get_num_comp_chunks()>=num_requested_comp_chunks)
                get_next_comp_chunk();
        }
}
```

7.1.2.1.8 Processor

The Processor object requests work to be done from the tracker as well as managing requests and replies for original data pieces to match the allocated work.

```
void Processor::run(){

        for (;;) {
                if ((!processing_finished()) &&
                  (parent->tracker->get_next_chunk_name(current_chunk_name))) {
                   // do we have this chunk?
                   if (parent->comp_torrent>orig_data_exists(current_chunk_name){
                        process_orig_chunk(current_chunk_name);
                   } else {
                // we don't have this chunk -
                // we are going to need to ask for it and wait
                parent->add_required_orig_chunk_front(current_chunk_name);
...
```

Here the node will grab a few work units near it as well to lessen the chance of starving again.

```
...
                parent->add_required_orig_chunk(child->GetText());
        }

        // if we're waiting on an original chunk to turn up
        // lets see if we've got it yet
        if (state == waiting_on_origchunk) {
          // have we got it yet? If so lets process it
          if (parent->comp_torrent->orig_data_exists(current_chunk_name)) {
                process_orig_chunk(current_chunk_name);
          } else {
                wait_timeout--;
...
```

Here there is a timeout process where if a work unit can't be found in a given period of time the node can tell the tracker to give it to someone else.

```
...
        }
```

7.1.2.2 Communication Protocol Between Nodes

Nodes communicate using a simple message based protocol using XML. Some messages are shared through limited broadcast where others are exchanged between connected nodes only.

<comptorrent> <packet>{welcome,denied} </packet> <ip></ip> <port></port> <uuid></uuid> <compchunks> <datachunk></datachunk> ... </compchunks> <origchunks> <datachunk></datachunk> ... </origchunks> </comptorrent> </comptorrent>	**Purpose:** During a connection negotiation, a welcome packet is sent by a server to the client on an accepted connect. On a refused connection, denied is sent instead with the rest of the information potentially omitted. **Parameters:** ip – The client's IPv4 address. port – The client's listening port. uuid – The client's unique id. compchunks – filenames of the parts of the original and computed data sets that this node possesses.
<comptorrent> <packet>connect</packet> <ip></ip> <port></port> <uuid></uuid> <routeid></routeid> </comptorrent>	**Purpose:** During connection negotiation, a connect packet is sent from the client to the server. **Parameters:** ip – The client's IPv4 address. port – The client's listening port. uuid – The client's unique id.

	routeid – The client's overlay key. Left blank if the client has not yet joined the overlay network.
<comptorrent> <packet>noop</packet> <ip></ip> <port></port> <uuid></uuid> </comptorrent>	**Purpose:** Sent periodically at either end of the connection to keep the connection active. **Parameters:** ip – The client's IPv4 address. port – The client's listening port. uuid – The client's unique id.
<comptorrent> <method>direct</method> <packet>file_req</packet> <dest></dest> <from></from> <type>{comp_data, orig_data}</type> <filehash></filehash> <name></name> <node_id></node_id> <ip></ip> <port></port> </comptorrent>	**Purpose:** Request a file to be routed around the network or from a directly connected IP connection. **Parameters:** method (direct) – Indicates that the packet is sent directly on a single IP connection and shouldn't be routed on. dest – The overlay route key of the destination. from – The overlay route key of the originating node. type – The data type computed or original. filehash – The md5 hash of the requested file. name – The name of the data requested. node_id - The client's unique id. ip – The client's IPv4 address. port – The client's listening port.
<comptorrent> <packet>file_update</packet> <filehash></filehash> <name></name> <type>{comp_data, orig_data}</type> <ip></ip> <port></port> </comptorrent>	**Purpose:** Sent between directly connected nodes to advertise an available file on a particular connection. **Parameters:** ip – The sender's IPv4 address. port – The sender's listening port. filehash – The md5 hash of the available file. type – The data type computed or original. name – The name of the available file.
<comptorrent> <packet>file_reply</packet> <dest></dest> <type></type> <ip></ip> <port></port> <filehash></filehash> <name></name> <data></data> </comptorrent>	**Purpose:** File reply packet. **Parameters:** ip – The sender's IPv4 address. port – The sender's listening port. dest – The overlay route key of the destination. type – The data type computed or original. ip – The sender's IPv4 address. port – The sender's listening port. filehash – The md5 hash of the available file. name – The name of the available file. Data – The file data. Either plain text or binary encoded base64 uuencoded text.
<comptorrent> <packet>overlay_update </packet> <routeid></routeid> <ip></ip> <port></port> </comptorrent>	**Purpose:** To advise connected nodes of a node's overlay key. **Parameters:** routeid – The overlay key for the node. ip – The sender's IPv4 address. port – The sender's listening port.
<comptorrent> <packet>overlay</packet> <routeid></routeid> </comptorrent>	**Purpose:** To allocate a directly connected node an overlay key. **Parameters:** routeid – The overlay key for the node.

Table 44: The message schema of the CompTorrent protocol.

7.1.2.3 Network Overlay & Topology

CompTorrent relies on a tree to organise its nodes into an overlay network. A simple key scheme is used where keys are allocated as a string derived from the node where the connection is being made. The "server" node allocates a string which is a concatenation of it's route key and a number representing the number of currently connected "client" nodes plus one. The seed node, and root of the tree, starts with a route of "1". If the root has two children they will have route keys of "11" and "12". If the right most node has a child it will have a key of "121", and so on. This arrangement is known as a trie and can be found described in Knuth (1997).

When a node wishes to make a new connection, they can do so in two ways. With the aid of the tracker, a connection candidate can be suggested for the new node. This can be based on an algorithm to moderate the nature of the overlay network. At present, nodes can be allocated randomly or be balanced by number of existing connections. In the case of the latter, existing nodes in the overlay network can be selected based on them having the least number of existing connections. Where the tracker is unavailable, a node can attempt a connection to another arbitrary known node (*bootstrapping* similar to Gnutella). In either case is the decision of the connection is made by the individual node, the tracker only assists the overall process it does not actively manage it.

Each node is allocated their route key and all of their child nodes will have keys that are a superset of their own key. When routing a message from one node to the other, messages can be sent from one node to any other in the network by traversing the tree upwards to where there is a match between the current node and the destination key. Worst case this means that a message will need to route all the way to the root before heading back down towards the destination. In practice a node will maintain more than one connection. It will usually maintain connections to its child nodes where it has previously allocated them a route key in the overlay network. Once a node has been allocated a place in the network it will make some connections over time, randomly or as directed by the tracker, to other nodes in various parts of the network. This allows messages to "short cut" through the overlay network. A message when it arrives will be routed on to its nearest neighbour as connected to that node – this will include its "worst case upwards" as well as a selection of potentially nearer nodes in the overlay.

This scheme has several advantages and disadvantages. The most obvious as already implied is that as long as you maintain a connection to a "higher" position in the overlay tree it is likely that you will be able to route a message. Implementation is simple with the core of the scheme implemented in only a few tens of lines of code. Decentralisation is also maintained by any node being able to allocate a routeable key without knowledge of the overall network. This adds the benefit of being robust in the face of access to a tracker and further decentralisation of swarm management. Separate trackers can manage node allocation and work unit management or multiples doing both.

As with most tree structures, worst case performance can see the nodes allocated into a list or a star. Also, with node churn it is possible for parts of the overlay to become unreachable. Both of these disadvantages are mitigated by the tracker paradigm and random node connections after overlay placement.

In addition, the hierarchical network scheme also lends itself well to the nature of computation. It is reasonable to assume that a seed is one of the most interested in the results of participating nodes in a computing swarm. Results moving

upwards towards the seed are naturally suitable. As are passing results around as one of the biggest incentives, in general, is an interest in either or both computed and original data sets. Bandwidth consumption is limited by the random allocation of computing jobs, and therefore original and computed data subsets, being spread around the overlay network reasonably evenly. Nodes can eavesdrop on moving messages, especially those requesting and replying to data sets, and make notes of likely nodes with data that they will later need or save data that they want as it passes through the network. Whilst the organisation of the network is hierarchical in a communication sense, there is no imposed hierarchy on the nodes themselves. Nodes can leave and join whenever and wherever they choose without disrupting computation or communication.

7.1.2.4 Security

In a distributed wide area volunteer network computing system, the integrity of the results and the confidence that these are results are correct is of paramount importance. In CompTorrent, as with most of these systems, confidence in computed data is gained through multiple computation of work units. The metadata files can also be digitally signed to allow a node to check that the metadata file is originally from a trusted source. Even a simple scheme of publishing a hash generated from the metadata file on a group controlled web site would afford a reasonable amount of certainty for users. This trust could conceivably increase if the hash matches on multiple unrelated and reputable websites.

That all being said a user still has to trust the seed and the tracker. The fact that metadata files contain executable code is cause for concern enough. In production, this concern can be mitigated by having metadata files digitally signed, as just discussed, by a seed and provided to the tracker. However, this only shifts the trust - it doesn't create it. A participant will need to trust that the project is as it is advertised. This trust would come through the number of nodes participating in the network, the project being associated with a trusted website (such as a faculty, company or organisation site) which would have a copy of the metadata signature on it as well. The metadata file, once trusted, allows that trust to be extended to the data sets as it contains hashes of every part of the original data set.

The tracker maintains security by keeping things as opaque as possible. It always opts for “blind requests” where possible e.g. “is this node linked to this node” rather than: “tell me all of the node connections that you know about”.

However, there are still many places where malfeasance can occur since CompTorrent has not been widely released. That is reasonable, at this stage, as CompTorrent is still “academic” code. Hardening needs to occur before it could be reliably used in production and is considered later in this thesis as further work.

7.2 The CompTorrent Metadata File

Building on the description given in 6.2.1, this section will describe each part of the metadata file in detail. The file presented is an abbreviated version of the CompTorrent file used in the Mandelbrot experiment presented later in 8.1.3.

```
<?xml version="1.0" ?>
<comptorrent>
<version>0.1</version>
<tracker_url>192.168.1.5</tracker_url>
<tracker_port>60000</tracker_port>
```

```
<name>mandelbrot</name>
<size>12345678</size>
<max_chunk_size>264000</max_chunk_size>
<md5>12DFF226FD2430A0F36204CD66423122</md5>
<algorithm>
<execution>comptorrents/working/mandelbrot/mandelbrot</execution>
<java_bytecode>
begin-base64 755 mandelbrot
f0VMRgEBAQAAAAAAAAAAAAIAAwABAAAAIIkECDQAAACwwwAAAAAAADQAIAAI
...
====
</java_bytecode>
<classname>mandelbrot</classname>
</algorithm>
<orig_data>
<file><name>mandelbrot_00000001</name><size>67</size><md5>12DFF226FD2430A0F36204CD664
23122</md5></file>
...
<file><name>mandelbrot_00000160</name><size>67</size><md5>12DFF226FD2430A0F36204CD664
23122</md5></file>
</orig_data>
</comptorrent>
```

Each element will now be considered in turn.

The version element allows for versioning of the system. This can allow the tracker and application know whether the metadata file is compatible with themselves before processing. The tracker URL and port indicates which tracker is the “home” tracker for this metadata file. It is possible to stipulate a different tracker and port when the CompTorrent application starts which will override the one given here as an option to allow trackers to move yet still capitalise on earlier metadata file distribution.

```
<version>0.1</version>
<tracker_url>192.168.1.5</tracker_url>
<tracker_port>60000</tracker_port>
```

The name and size of the computing project is given here. The name should be human readable and relatively unique (to the hosting tracker at least). The size refers to the total size of the original dataset for computation.

```
<name>mandelbrot</name>
<size>12345678</size>
```

The max_chunk size refers to the largest piece that the original data set has been broken into. This allows for memory allocation in the application but also gives an indication of the size of each work unit. The md5 hash here is the hash of the entire original dataset. This allows an application to verify the entire set should it be obtained.

```
<max_chunk_size>264000</max_chunk_size>
<md5>12DFF226FD2430A0F36204CD66423122</md5>
```

The algorithm section contains two parts; the execution and bytecode. The execution element tells CompTorrent how the embedded algorithm is to be executed. It can be a script, or as in the example here, a single command. The bytecode section includes a base64 encoded copy of the algorithm. This can be java, as it is in this case, or an executable. Its possible that the embedded algorithm can simply be a downloader for another larger or more frequently changing algorithm that isn't suitable for embedding into the metadata file itself.

```
<algorithm>
<execution>comptorrents/working/mandelbrot/mandelbrot</execution>
<java_bytecode>
begin-base64 755 mandelbrot
f0VMRgEBAQAAAAAAAAAAAAIAAwABAAAAIIkECDQAAACwwwAAAAAAADQAIAAI
...
====
</java_bytecode>
<classname>mandelbrot</classname>
</algorithm>
```

The original data section describes every subset of the data which forms the basis of each work unit. Each "chunk" of the data has a name, a size and a computed md5 hash. The name and the hash are used as a way of identifying and verifying data received from nodes on the network respectively.

```
<orig_data>
<file><name>mandelbrot_00000001</name><size>67</size><md5>12DFF226FD2430A0F36204CD664
23122</md5></file>
...
<file><name>mandelbrot_00000160</name><size>67</size><md5>12DFF226FD2430A0F36204CD664
23122</md5></file>
</orig_data>
```

7.3 CompTorrent in Action

Now that the major components of the system are understood, a "step through" of the processes are presented here to show how the system functions as a whole. Typical states are given and discussed.

7.3.1 A Seed Node by Itself

A computing exercise will always start with a single seed and a CompTorrent file. The same CompTorrent file is used by the seed to start the computation as well as it being distributed to other potential nodes to join later.

The seed's actions at start are mostly managed by the main loop itself:

1. Set my route id to 1 and contact the tracker and register the project.
2. Create an Controller object to manage the runtime operation of the application. Here Controller will create a Processor object which, depending on the parameters given to the seed at start up, may start working on the computation itself. This is an option as some seeds may wish to merely upload the source data as quickly as possible and then disconnect.
3. Parse the CompTorrent file, extract the algorithm and create working directories.
4. Create a CompTorrentPeer object to manage messages.
5. Start a listener object to accept incoming connections.
6. Enter the main loop and repeatedly try to make outgoing connections to other nodes as they become known to the seed.

7.3.2 A Node Joining Without a Route ID

New nodes will make a connection to an existing node without having an overlay key or route id. This means that one is allocated to them as part of the connection process. From the existing node on the network's perspective (the server in the case of this direct connect), this is an incoming or client connection.

1. A new node will contact the tracker and receive a suggestion for a node to connect to.
2. Through Controller.attempt_outgoing_connection() the node will create a new Controller object for this connection and start it.
3. Controller will create a CompTorrentPeer object that will go through the connection process and send a connect message. If it gets a welcome message it will add itself to a collection of connections and notify the tracker that a connection has been made. If it is refused it will simply terminate itself.
4. At the server side of this new connection, the corresponding CompTorrentPeer object (the same object is used for client and server connections) should observe that the connecting node doesn't have a route id and will derive one from its own route and send an overlay message directly to the new node. It is perfectly possible for multiple connection to be occuring concurrently so it is the first received overlay message that is accepted by the new node and its route key that is adopted as its own. Further overlay messages are ignored if the node already has a route id. Once this has occurred the new node will broadcast its overlay key to directly connected nodes as well as informing the tracker.
5. Now that the node is registered on the overlay network, its processor object should begin to request and process work units.

7.3.3 A Node Joining a Node

As a part of the main loop of the CompTorrent application, periodically nodes will attempt to make connections to other nodes on the network, even though they already have a primary connection from when they were allocated a route id. These are outgoing connections that the node makes to improve the robustness and performance of the network.

This process is identical to steps 1,2 and 3 just described in 7.3.2. Obviously steps 4 and 5 are related to obtaining a new route key. As this node is already a part of the overlay network, it will furnish the server node with its overlay id during the connection request. On successful connection, both nodes will notify the tracker that a direct, or ip, connection has been made.

7.3.4 Requesting & Completing a Work Unit

Work units are requested and completed in the Processor object.

1. Providing the node has a route id (as it it pointless trying to do computation if you won't be able to get files) and that the processing has not been deemed finished, the Processor will contact the tracker and request a work unit.
2. When a work unit is allocated, the processor checks to see if it already has

the matching original data file, if so it begins processing the work unit. If not, it will pass a request for the file to the Controller object which will then pass the request out to its connected nodes via a tracker hint if it doesn't already have a directly connected node with the required file. The processor will then wait for the file to arrive before starting processing. Obviously, in the simplest case described, this provides much scope for starvation. However, this is mitigated by the tracker suggesting work units based on files that the node already has (as a result of eavesdropping on passing through requests and requesting requesting more than one original data file at a time) or files that are known to be nearby.

3. When the processing has completed, the node contacts the tracker and lets it know that its been done and what the calculated hash of the result is. The tracker can then verify this result by allocating the same work unit to different nodes.

7.3.5 Verification of the Work Unit

The actual verification process is identical to requesting a work unit. A tracker suggests work units and the nodes don't really know if they have been computed already or not. A mischievous or particularly attentive node may be able to measure how long work units stay in an allocated state, but this is difficult with the variation of time taken for each work unit to be completed – either due to the nature of data or to the variation of network and processing speed.

Once the tracker receives a number of resulting computed data sets with matching hashes, some confidence in the correctness of the computed work can be had. Should conflicting results be received, the latest received work unit is discarded. Where there are only two results, the newest work unit replaces the original one to avoid the scenario of a first result being incorrect .

7.3.6 Assembling the Results

In the simplest case, the original and computed datasets can be reassembled through concatenation. A suggested extension of the system would involve adding another element to the metadata which would contain a script for post-processing of the dataset on completion.

7.4 Summary

This chapter has described the major components of the CompTorrent system, the Tracker, Metadata file and CompTorrent Application, in technical detail. Discussion of design considerations was also presented along with discoveries and compromises made along the way.

In the following chapter, an evaluation of this system is presented along with a comparison to the traditional distributed computing platforms as presented in Chapter 5.

Chapter Eight

8.0 CompTorrent Evaluation

In this Chapter an evaluation is presented of the system described in Chapters 6 and 7 using WAN-DC, the benchmark presented in Chapter 4. These data were gathered on the same hardware as the evaluation of BOINC and Condor as presented in Chapter 5 and as such, allow us to compare these 3 different systems as fairly as possible. In this chapter it will be demonstrated that a peer-to-peer distributed computing system can produce similar performance results to a traditional client/server distributed computing system.

8.1 Performance Results

In this section the performance results are presented for CompTorrent with discussion of the results at each step.

8.1.1 POV-Ray

CompTorrent performed well in the *POV-Ray* experiment showing conventional results across the range of participating node sizes with speedup efficiency remaining between 0.96 and 0.98.

CompTorrent POV-Ray				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	16348	64.08	1	1.00
2	8521.8	91.8	1.92	0.96
4	4250.4	62.2	3.85	0.96
8	2089	3.16	7.83	0.98
16	1054	36.96	15.51	0.97

Table 45: CompTorrent results for the POV-Ray experiment.

8.1.2 Transcode

Transcode performed relatively poorly with speedup peaking at 8 machines. The intrinsic overhead of the transcode application has a marked effect on the amount of speedup that can theoretically be achieved.

CompTorrent Transcode				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	1091	6.63	1	1
2	602.6	45.54	1.81	0.91
4	352.2	18.39	3.1	0.78
8	202.2	21.63	5.4	0.68
16	236.4	18.45	4.62	0.29

Table 46: CompTorrent results for the Transcode experiment.

8.1.3 Mandelbrot

CompTorrent produces lacklustre results for *Mandelbrot.* Speedup peaks at 8 machines with little change in efficiency across the range with a maxima of only 2.41 and an efficiency of around 30%.

CompTorrent Mandelbrot				
CompTorrent	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	444.8	83.42	1	1
2	240	2.35	1.85	0.93
4	203.8	23.86	2.18	0.55
8	184.8	18.46	2.41	0.3
16	234.4	25.81	1.9	0.12

Table 47: CompTorrent results for the Mandelbrot experiment.

8.1.4 No Work

No Work illustrates the bare overhead of CompTorrent and what kind of speedup can be observed with jobs of no time whatsoever. It peaks at 8 nodes and begins to reduce by 16. Some similarity is observed between the results here and that of Mandelbrot, with the peak at 8 machines and a similar speedup at 2 machines.

CompTorrent No Work				
Num Machines	**Mean Run (secs)**	**Standard Deviation (secs)**	**Speedup**	**Speedup Efficiency**
1	103.8	1.1	1	1
2	58.8	4.21	1.77	0.89
4	40	1.87	2.6	0.65
8	34.4	1.34	3.02	0.38
16	36.8	3.7	2.82	0.18

Table 48: CompTorrent results for the No Work experiment.

8.1.5 One Second

Figure 24 shows that two machines, whilst quite erratic in places, are not much worse in efficiency at the corresponding job length for any of the large node sizes. And where the measurements appear erratic is where they are swinging toward 100% efficiency (expressed as known processing time divided by wall clock time), so whilst less deterministic, it is in the favour of overall job completion time being lower.

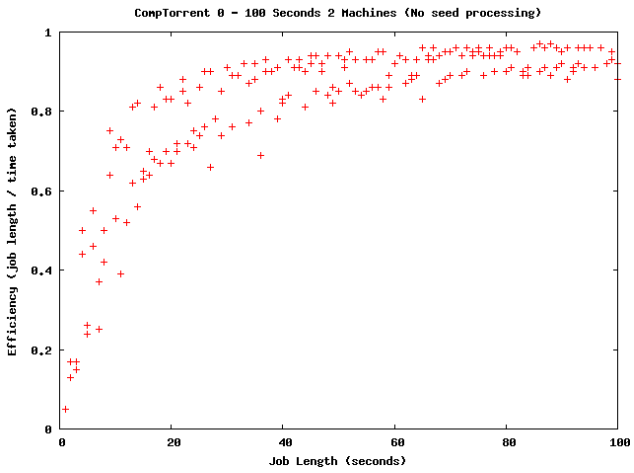


Figure 24: CompTorrent One Second graph for 2 machines.

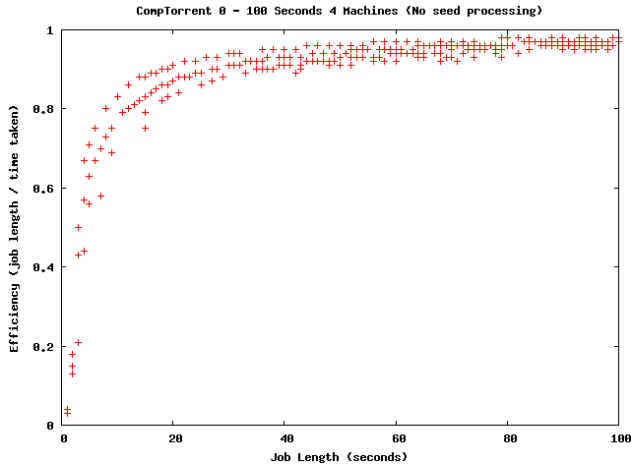


Figure 25: CompTorrent One Second graph for 4 machines.

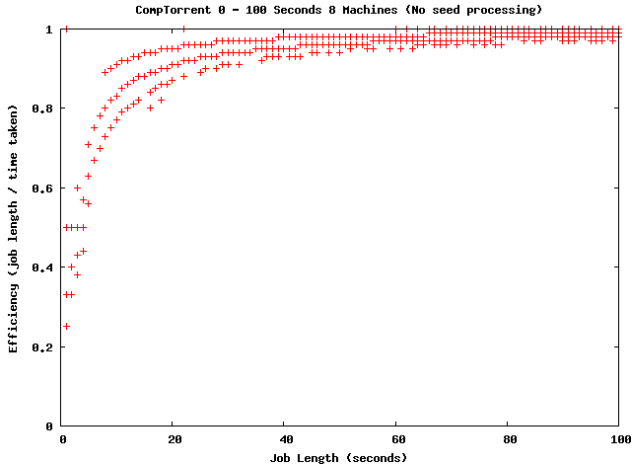


Figure 26: CompTorrent One Second graph for 8 Machines.

4, 8 and 16 machines show an obvious curve where jobs of around 20 seconds in length become acceptable for the system as configured.

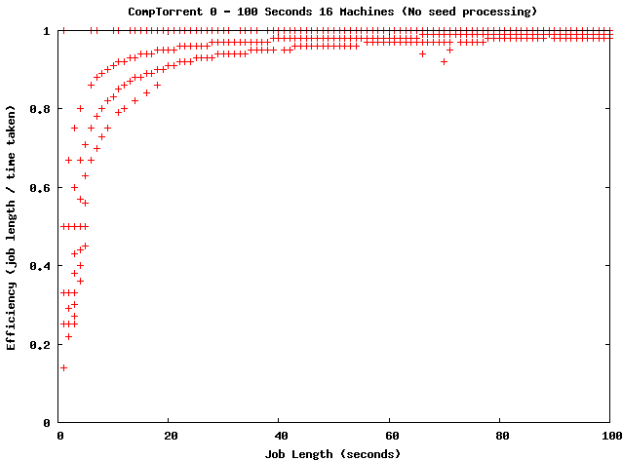


Figure 27: CompTorrent One Second graph for 16 machines.

Figure 28 shows all of the previous One Second results as a mean for 1,2,4,8 and 16 node cluster sizes respectively.

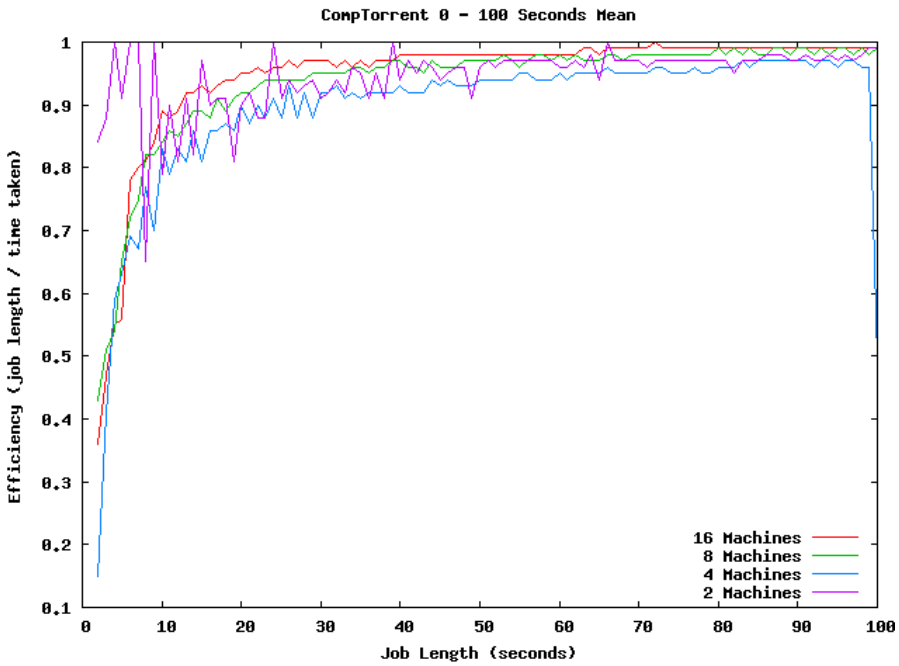


Figure 28: CompTorrent Mean One Second results.

8.1.6 Mean Work Unit

Table 49 shows the mean work unit results for POV-Ray, Mandelbrot, Transcode and No Work. Similarly to Condor presented earlier, the most obvious thing it shows is the difference in processing times between the different experiments. Mandelbrot, Transcode and No Work, when compared to POV-Ray, are comparatively light in their processing time per work unit for the same reasons. That is, the work units of POV-Ray are differing in their load and that CompTorrent is relatively efficient in processing work units to allow this to so obviously show.

Mandelbrot also displays a large difference between individual runtimes for a single machine. Care must be taken with varying work unit algorithms like Mandelbrot on a small number of machines. Since Mean Work Unit looks for the average time to get a response with a computed data set, the difference between time taken for each work unit will vary on a single machine as the variance is being driven by the nature of the different work sizes and not the network or the distributed computing system's internal overhead. This is shown obviously with a higher deviation than mean result for Mandelbrot with 1 machine in Table 49.

Experiment Name	Num Machines	Mean (secs)	Standard Deviation (secs)
POV-Ray	1	259.49	46.17
	2	135.27	79.21
	4	67.47	46.47
	8	33.16	29.67
	16	16.73	14.13
Mandelbrot	1	2.8	3.16
	2	1.51	0.79
	4	1.28	0.47
	8	1.16	0.39
	16	1.47	0.59
Transcode	1	11.02	1.49
	2	6.09	2.72
	4	3.56	1.40
	8	2.04	0.74
	16	2.39	1.03
No Work	1	1.05	0.20
	2	0.59	0.25
	4	0.4	0.34
	8	0.35	0.51
	16	0.37	1.23

Table 49: Mean Work Unit results for CompTorrent

8.2 Qualitative Results

APPROACH / DESIGN	
	CompTorrent
Approach	Volunteer / Cluster
Node Organisation	Peer-to-peer
Network topology	Ad Hoc
Application	Task
Requirements & Dedication	Idle cycles or greater
FEATURES	
	CompTorrent
Algorithmic Suitability	No interprocess communication
Standards supported	None explicitly
Hardware / OS support	***Platform Independence:*** Unix, Linux, Mac. Can be compiled for Windows with little effort. ***Hardware:*** As supported by the above operating systems. ***GPU Processing Support:*** No. ***Heterogeneous:*** Any hardware supported by the operating systems above.
TASK & RESOURCE MANAGEMENT	
	CompTorrent
Resource allocation to Jobs	Server or Node.
Task allocation	Server or Node.
ROBUSTNESS	
	CompTorrent
Checkpointing	No.
Scalability	Experimentation only.
Quality of service	No
Churn	Yes
Malfeasance	Multiple calculations
LICENSING	
	CompTorrent
Licence Description	GNU Lesser General Public License
Source Availability	Open
Governing Organisation	Individual
USABILITY	
	CompTorrent
Hosting a new project	A originating node or any size. A tracker which can be hosted on the originating node or elsewhere.

Joining an exiting project	Download software and pick a project.
Coding for a new project	None required. Can use existing binaries.
SUPPORT	
	CompTorrent
Commercial	No
Community	Web site and mailing list.
Books	No. PhD thesis.
PARTICIPATORY INCENTIVES	
	CompTorrent
Data Access	Yes. Source and computed data set. Could be limited by the project but it would defeat the purpose of the system.
Financial through statistics	Possibly – could be controlled by a tracker.
"Fame" through statistics	Yes – by tracker.

Table 50: WAN-DC qualitative results for CompTorrent.

8.3 Discussion of Results Compared with BOINC & Condor

As alluded to in the introduction, a variety of systems have been chosen to represent several of the different approaches to distributed computing that present a reasonable survey at time of writing. The results obtained for CompTorrent, as just described, will be compared to the results described in Chapter 5 for BOINC and Condor.

POV-Ray Speedup			
Cluster Size	**Condor**	**BOINC**	**CompTorrent**
1	1	1	1
2	1.62	1.95	1.92
4	3.86	3.93	3.85
8	7.74	8.34	7.83
16	15.66	18.89	15.51

Table 51: POV-Ray speedup results for the three systems.

In Table 51, CompTorrent compares favourably with both BOINC and Condor with all systems producing similar results across the range of cluster sizes. In cases 1, 2 and 8 the results of CompTorrent fall between those of BOINC and Condor. In the case of 4 and 16 machines, CompTorrent is worse that both Condor and BOINC though only by a small margin. Super-linear speedup was observed for BOINC (from hyper-threading as discussed previously) in the cases of 8 and 16 machines however all results, for all systems, remain close to ideal across the range of cluster configurations. This is due to the desirable processing (much) and communication (not much) ratio of the POV-Ray algorithm.

Transcode Speedup			
Cluster Size	**Condor**	**BOINC**	**CompTorrent**
1	1	1	1
2	3.89	1.59	1.81
4	4.87	3.29	3.1
8	5.39	3.81	5.4
16	5.45	5.51	4.62

Table 52: Transcode speedup results for the three systems.

Transcode is presented in Table 52, showing results for CompTorrent that are either between the range of BOINC and Condor (for 2 and 8 machines) and worse for 16 and 4 machines. Transcode, being a video processing experiment, has a combination of high network load as well as high computation.

Mandelbrot Speedup			
Cluster Size	**Condor**	**BOINC**	**CompTorrent**
1	1	1	1
2	1.45	1.88	1.85
4	1.83	3.43	2.18
8	1.86	5.07	2.41
16	1.84	6.8	1.9

Table 53: Mandelbrot results for the three systems.

In Mandelbrot (Table 53), CompTorrent demonstrates results similar to that of Condor with marginally more speedup in each case. BOINC shows much better speedup results however, referring back to Section 5.4.2.3, this is due to very long runtimes for this experiment compared to Condor and CompTorrent.

No Work Speedup			
Cluster Size	**Condor**	**BOINC**	**CompTorrent**
1	1	1	1
2	1.62	1.87	1.77
4	1.62	2.37	2.6
8	1.64	3.43	3.02
16	1.65	3.3	2.82

Table 54: No Work results for the three systems.

Table 54 shows CompTorrent again producing results between both Condor and BOINC, with the exception at 4 machines, where CompTorrent performs marginally better than either system.

For the Mean time for a work unit to be returned, Tables 55,56,57 and 58 all show CompTorrent behaving between the ranges of, or better than, Condor and BOINC.

Mean Work Unit - Transcode			
Cluster Size	**Condor (secs)**	**BOINC (secs)**	**CompTorrent (secs)**
1	11.74	26.46	11.02
2	3.02	16.64	6.09
4	2.41	8.03	3.56
8	2.18	6.95	2.04
16	2.15	4.8	2.39

Table 55: Mean work unit (Transcode) results for the three systems.

Mean Work Unit - Mandelbrot			
Cluster Size	**Condor (secs)**	**BOINC (secs)**	**CompTorrent (secs)**
1	3.84	16.9	2.8
2	2.64	9	1.51
4	2.10	4.92	1.28
8	2.07	3.33	1.16
16	2.09	2.48	1.47

Table 56: Mean work unit (Mandelbrot) results for the three systems.

Mean Work Unit - POV-Ray			
Cluster Size	**Condor (secs)**	**BOINC (secs)**	**CompTorrent (secs)**
1	257.43	267.61	259.49
2	159.09	137.36	135.27
4	66.67	68.12	67.47
8	33.24	32.13	33.16
16	16.43	14.15	16.73

Table 57: Mean work unit (POV-Ray) results for the three systems.

Mean Work Unit – No Work			
Cluster Size	**Condor (secs)**	**BOINC (secs)**	**CompTorrent (secs)**
1	3.49	16.75	1.05
2	2.16	8.94	0.59
4	2.16	7.06	0.4
8	2.13	4.89	0.35
16	2.11	5.08	0.37

Table 58: Mean work unit (No Work) results for the three systems.

8.4 Chapter Summary

This chapter has presented and discussed the results for CompTorrent and compared them to the results gathered earlier for BOINC and Condor. In all performance

metrics, CompTorrent has presented with similar results to BOINC or Condor whilst having a completely different underlying peer-to-peer architecture. This peer-to-peer system, as presented and under the described conditions and benchmarks, has produced comparable results to two traditional distributed computing systems. This has shown, within the limits of the systems tested, that a peer-to-peer approach to be an acceptable one for distributed computing.

Chapter Nine

9.0 Conclusions & Further Work

This chapter summarises how the research presented in this thesis has achieved its stated goals. It also discusses future research directions that have been identified and concludes with some closing remarks about the nature of peer-to-peer as a topic of research at time of writing.

9.1 Conclusions

This thesis primarily considered the problem of the cost of hosting a distributed computing project in a traditional manner. From this the research questions arose: Can peer-to-peer techniques be applied, which are known to have a lower project ownership cost, to general purpose distributed computing effectively? And, how can this new system be measured against other existing distributed computing systems?

This section will summarise the answers to these questions that this thesis has answered.

9.1.1 Distributed Computing with Peer-to-Peer Computing

This thesis has shown that using peer-to-peer techniques with distributed computing is not only feasible, it shows that it compares well with existing client-server approaches in terms of the computing performance achieved as well as the levels of service offered.

9.1.1.1 Comparative Performance and Scalability

According to the WAN-DC benchmark results show that CompTorrent maintains a performance range between that of BOINC and Condor for all cluster node sizes for the POV-Ray experiment. Transcode shows CompTorrent is between or better than BOINC and Condor in 50% of cases and, whilst worse in the other half of experiments, it was only by approximately 15% in the worst case. Mandelbrot showed results between both distributed systems and, similarly to POV-Ray, No Work behaves either between or better than the results of both BOINC and Condor.

Whilst limited by the smaller scale scenario presented within this thesis, these results from Chapters 5 and 8 show that that a distributed processing system based on a decentralised, peer-to-peer network can provide similar results to distributed processing systems based on traditional client/server networking architectures. This has been proven on hardware under controlled conditions without the need to resort to simulation.

9.1.2 Ease of Use

As discussed in Chapter 6, CompTorrent requires very little effort to install and use – from either the user or project manager's point of view. Once the project manager has identified the algorithm to be used, packaged it within a script in such as way that it has a data in and data out file-based interface, the job is almost done - only the

generation of a metadata file and associated work units is all that is required. This level of work is also generally true of BOINC and Condor to this point, however the next step is key – no server requirements. A tracker must be used of course, but this does not necessarily need to be under the control of the project owner. A seed is then started and the project begins. Indeed, once the original dataset has be uploaded to the swarm it is not necessary for it to continue to remain a part of the network. Later it could reconnect as a normal node and retrieve the computed dataset just like anyone else.

It is hoped that future development of this project will allow more economies to be found in the instigation and maintenance of distributed computing projects.

9.1.3 New Incentives, Network Time, Processing Time and Machine Dedication

If CompTorrent's operation is considered, obviously the time taken to upload the full data set to the swarm versus the time taken to compute the result set will be paramount for the seed when considering the value of a CompTorrent exercise. This relates strongly to what is classically considered to be the efficiency metric in distributed computing where efficiency is measured as computation versus communication. Some more trivial applications of CompTorrent, such as recoding video, may take longer to upload and download the original and computed sets respectively and therefore yield no computation advantage at all. However, the overall time taken for the swarm as a whole to receive a computed set is determined by the upload speed of the original seed and the algorithm run time. So, if the swarm as a whole is interested only in the computed set, simultaneous computation and distribution may well get each node the computed set in less overall elapsed time when compared to single machine computation first and then distribution. Certainly the lag between the original data set becoming available and the start of the computed set being distributed is minimized with CompTorrent. One needs to keep in mind here that this is public computing where many participants are interested in the data sets. Obviously in a private computing exercise these mitigating circumstances, to what would otherwise be a waste of time, would not apply.

There are other incentives for a seed over just elapsed time as well. The load placed on the machine itself is a factor when the primary consideration is low-cost ownership of hosting the project. The time taken for a single processor to complete a computation job obviously relies on the load placed on the machine itself. It may be reasonable to recode 4.6Gb of MPEG1 video to 700Mb of MPEG4 in 4 hours providing the machine is absolutely dedicated to the task. An 8 hour simultaneous upload, compute and download may be preferable where the machine is not highly loaded by the computing task and so can be readily used for other work.

9.2 Further Work

Here further work identified during the course of this thesis is discussed for the major works within this thesis of WAN-DC and CompTorrent.

9.2.1 WAN-DC

Clearly, for wider utility, WAN-DC needs to have algorithms included which support inter-process communication. Also, the qualitative metrics could be improved significantly through survey work to allow for a more rigorous approach. However,

as alluded to previously in chapter four, this benchmark grew out of necessity rather than being an exercise of trying to be all things to all situations as this could easily grow into the scope of an entire thesis in itself. Here are presented some options for consideration for later expansion.

9.2.1.1 Performance Metrics

Two broad categories of inter-process communication should be considered for integration into WAN-DC. They are, of course, loosely and tightly coupled algorithms. An example of a loosely coupled application, with good prospects for scalability, would be calculating primes using the classical algorithm of the Sieve of Eratosthenes with strike out. Another loosely coupled example, with less natural scalability, could be calculating Fibonacci numbers. This algorithm's inherent sequential nature could lead to an investigation of how distributed systems deal with largely sequential problems in a wide area network distributed computing network with all the challenges it presents. Other problems should be identified to produce some examples between these two extremes.

In the case of tightly coupled problems there are a plethora of examples where communication overhead has the potential to overwhelm a system. Parallel sorting applications, such as Mergesort (Knuth 1998), are common examples, with implementations found readily in the literature. Matrix based applications, particularly wave front computations such as heat transfer, give a good, adjustable level of granularity based on the lengths of the communicating edges of the matrices shared between nodes.

Other additions to this benchmark could include testing for malfeasance, measuring robustness, further work in establishing baselines or seeing how degradation of the underlying network performance affects these systems. Several of these benchmarks were considered during this thesis work and have been included in Appendix D as a head start for future work in this area.

Integration of existing parallel benchmarks, as appropriate to wide area network distributed computing, could be incorporated. For example, determining where the NAS Parallel Benchmarks could apply to peer-to-peer systems would be useful.

9.2.1.2 Qualitative Metrics

Experimental design for survey work of the qualitative options as well as formally devising a method of obtaining results from a wide audience.

9.2.1.3 More Systems & Underlying Network Conditions

A framework's value is also increased through the number of machines and systems that have been evaluated and whose statistics are available. In this thesis, three systems were evaluated using WAN-DC. Extending this to other hardware and systems, in particular Grid and tuple space based systems would be of considerable value.

More work on formalising the conditions for applying the WAN-DC benchmark, and benchmarks in general, would be useful for modern distributed systems. At present it is difficult to quantify the nature of the network that the system is on (one of the main reasons this thesis's results concentrated on a controllable network environment). Developing a method for the approximate quantification of a

set of nodes participating in a swarm at a given time will greatly increase the comparability of benchmarks being applied to these wide area network systems. System results will have greater value if the underlying network conditions are understood at the time.

Also, the limited and small nature of the cluster employed, is an obvious candidate for future work. With more resources, beyond that available to this project, it would be possible to construct a stable WAN cluster with which to repeat this experimentation to see how the systems compare over a larger geographical scale.

9.2.2 CompTorrent

CompTorrent has demonstrated the contributions of this thesis however there is always scope for further extension and improvement. This section will briefly discuss some suggestion for further work with CompTorrent.

9.2.2.1 Protocol and Routing Optimisation

The usage of XML for communication between nodes for all communications could be wasteful for fine-grained tasks. XML was originally chosen due to its ease of extensibility and modification as the research progressed. Beyond compressing the XML, which is a possibility but again unlikely to yield much gain for very small payloads (Goldsmith 2004), a binary structure implementation where the bare minimum of meta data is exchanged is feasible. A solution between binary structures and XML that has gained popularity during the progress of this thesis is YAML, which provides a lighter weight mark up language for structured data (Ingerson, Evans & Ben-Kiki 2001).

Examining the relationships between the nature of the computation task and the topology of the overlay network is already showing promise. Applying different routing algorithms is an area in its own right and further work beyond the least common ancestor heuristic which is used now should prove worthwhile. Other routing arrangements used in distributed hash tables such as a Skiplist, Cartesian Coordinate Space, Plaxton Tree and similar could be compared to see if they offer performance benefits whilst considering their cost in terms of implementation complexity and transparency for the user.

9.2.2.2 Interprocess Communication

Another obvious extension would be support for algorithms that are not completely independently parallel. The classic choices between shared memory or message passing are two obvious candidates for implementation and testing. Implementing shared memory across nodes in a CompTorrent swarm would also allow for a distributed tracker to be overlaid on the network. This could either be as a primary or secondary tracker service and it would be interesting to see how this could be used to improve the robustness of the system.

9.2.2.3 Optimisation of File Transfer

Optimization of file transfer is another area that may yield improved results. A lot of work has already been done investigating the efficiency of BitTorrent for file transfers including some recent work (Piatek et al 2007) that has further increased performance by some 70% by selective uploading to connected peers based on their behaviour. It will be interesting to see if these ranking algorithms would have a

similar result with peers based on their bandwidth contribution as well as their computing contribution. This would expand on existing work of allocating tasks based on the number of data chunks processed, number of file requests services, time taken to respond, etc.

9.2.2.4 Trackers

The tracker is currently an HTTP service and has a relatively small bandwidth load (subject to the granularity of the task and data). A recent idea involves investigating the possibility of embedding tracker data into unlikely places or protocols. As the tracker is mainly shared memory (lists of connected nodes, completed chunks) it may well be possible to host tracker data on another unrelated service such as Internet Relay Chat. Security related issues is also an area where much work can be done. An obfuscation technique that has already been proven in concept is embedding tracker data into an image using steganographic techniques. It will be interesting to see if the extra bandwidth required will result in any stealth advantage. As would looking at the mobility of projects between trackers during computation.

The tracker could obviously be extended to involve more computation when suggesting nodes for connection. It has been kept as simple as possible in the case of CompTorrent in order to prove that a genuinely decentralised swarm can produce similar results to a client-server system. In practice, especially where a higher confidence in nodes is recognised (i.e. in a controlled, production environment), more tracker involvement would be sensible. However, in a peer-to-peer system this should always be carefully considered along with the level of centralisation the system is to maintain.

9.2.3 Botnets

As previously mentioned in 3.3.3, botnets have been potentially identified as some of the most distributed, peer-to-peer computing systems known to be in practical, albeit illegal, use. Further work to identify existing techniques being used by these networks could compliment the work in this thesis as well as providing potential insight into mitigating the effects of these harmful botnets. This is raised here as a general research direction beyond the scope of CompTorrent and general purpose distributed computing.

9.3 Some Personal Concluding Remarks on Peer-to-Peer as a Controversial Research Topic

At the beginning of Chapter 1 of this thesis, a quote from J. Bronowski was included which referred to the nature of a scientist as being one of dissent. Peer-to-peer computing, as a technique, has come under great, sustained fire from the media, industry groups and politicians as being synonymous with illegal file sharing and the distribution of other prohibited content. There have been many calls for it to be banned and many attempts for it to be blocked at a Internet Service Provider level.

This incorrect assumption that a networking or computing technique equals malfeasance and corruption is one that must not be allowed to continue to propagate further. This is hardly the first time that computing has been controversial, yet I feel that it is significant for us here as so many different groups are in active opposition at once.

In this thesis it has been shown that peer-to-peer networking can provide a

useful harnessing of resources that might not otherwise have been economically feasible and therefore available to a research group. This allows scientific projects another avenue for procuring computing cycles just like BOINC and Condor (and many others) are doing now. This project, at time of writing, uniquely allows the not necessarily professional and less funded groups or individuals the ability to host a distributed computing project easily and virtually without cost.

In this field, researchers should find it their duty to show how peer-to-peer is not all about "MP3s and piracy" and continue to speak out at ill-informed debate and demonstrate where peer-to-peer as a discrete technology itself is being used for the real benefit of humanity. The mesh networking scheme used in the One Laptop Per Child project is a real example of peer-to-peer being a part of the success of a humanitarian effort.

It is hoped that this thesis, and others like it in the area of peer-to-peer computing, will show that this technique is overwhelmingly benign and can be used productively and widely for the benefit of all.

10.0 References

Aberer, K., 2001, *P-Grid: A Self-Organizing Access Structure for P2P Information Systems*, Proceedings of the Ninth International Conference of Cooperative Information Systems, 2001.

Amdahl, G., 1967. *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*, AFIPS Conference Proceedings, (30), pp. 483-485.

Anderson, D., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D. 2002. SETI@home: *An Experiment in Public Resource Computing*. Communications of the ACM. Vol. 45 Issue 11, (2002) 56 – 61.

Anderson, D. 2004. BOINC: *A system for public-resource computing and storage.* In 5th IEEE/ACM International Workshop on Grid Computing.

Apple Inc., 2007, *Xgrid Technical Brief*, <http://images.apple.com/server/macosx/docs/L355779B_Xgrid_tb.pdf>. Last Accessed 4 April 2009.

Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Venkatakrishnan, V., Weeratunga, S., Simon, H., 1991. *The NAS parallel benchmarks*. The International Journal of Supercomputer Applications.

Bakker, A., Kuz, I., Steen, M. 1997, *Towards a taxonomy of distributed-object models*, In Proceedings of the Third Annual ASCI Conference, pp. 22-27.

Bharambe, A., Herley, C., Padmanabhan, V. N. 2005. *Analyzing and Improving BitTorrent Performance*. Technical Report. MSR-TR-2005-03, Microsoft Research.

BOINC, 2007, *Boinc Project, Other Information on Creating BOINC projects*, <http://boinc.berkeley.edu/trac/wiki/OtherProjectDocs>, Last Accessed 18 December 2008.

BOINC, 2007, *Boinc Project, Recruiting and retaining volunteers*, <http://boinc.berkeley.edu/trac/wiki/VolunteerRecruit>, Last Accessed 3 April 2008.

Boklund, A., Mankefors-Christiernin, S., Jiresjö, C., Namaki, N., 2005. *COTS-Cluster Evolution during the Last Decade and Extrapolation into the Next*. Proceedings Parallel and Distributed Computing and Networks (Sweden).

Bratosin C., van der Aalst, W., Sidorova N. 2007. *Modeling Grid Workflows with Coloured Petri Nets*, Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, CPN'07, Aarhus, Denmark 2007.

Buyya, R. 2002. *Economic-based Distributed Resource Management and Scheduling for Grid Computing,* Ph.D. Thesis, Monash University, Melbourne Australia, April 12, 2002.

Buyya, R., Yeo, C., Venugopa, S. 2008. *Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities*. In Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08, IEEE CS Press, Los Alamitos,CA, USA).

Carriero, N., Gelernter, D. 2006. HPCWire. Letter to the Editor: *The Tuple Space Solution*. June 16, 2006. <http://www.hpcwire.com/topic/systems/17885239.html> Last Accessed 1/12/2008

Centre for Computing History, 2009. *Computing History – Convergent Technologies*. <http://www.computinghistory.org.uk/sec/2963/Convergent-Technologies/>. Last accessed 5 April 2009.

Chang H., Li, K., Lin, Y. Yang, C., Wang, H., Lee L,. 2005. *Performance issues of grid computing based on different architecture cluster computing platforms,* 19th International Conference on Advanced Information Networking and Applications, 2005. AINA 2005. Volume 2, pp 321 – 324.

Cortell, J. 2005, *Lecturer censored in Spanish University (UPV) for defending P2P networks*, <http://homepage.mac.com/jorgecortell/blogwavestudio/LH20041209105106/LHA20050520091532/index.html> Last Accessed 25 March 2009.

Costa, F., Silva, L., Fedak, G., Kelley, I., 2008. *Optimizing the data distribution layer of BOINC with BitTorrent*. IPDPS 2008 Parallel and Distributed Processing, 2008.

Curnow, H.J., Wichman, B.A. 1976. *A Synthetic Benchmark*, Computer Journal, Volume 19, Issue 1, February 1976., p. 43-49.

Dongarra, J., Luszczek, P., Petitet, A. 2003. *The LINPACK benchmark: Past, present, and future*. Concurrency and Computation: Practice and Experience 15, 1-18.

Dumas, E. 2001. *Programme de bench calcul de l'ensemble de Mandelbrot* . Available online: <http://magnux.free.fr/gcc/mandelbrot.c>. Last accessed 26 April 2009.

Eberspächer, J., Schollmeier, R. 2005, *Peer-to-Peer Systems and Applications: First and Second Generation of Peer-to-Peer Systems*, Lecture Notes in Computer Science, Springer.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., 1999, RFC2616: HTTP1/1 Specification, Network Working Group, IETF.

Flynn, M. 1972. *Some Computer Organizations and Their Effectiveness*, IEEE Transactions. Computer, Vol. C-21, pp. 948, 1972.

Foster, I., 2002, *What is the Grid? - a three point checklist*, *GRIDtoday*, Vol. 1, No. 6.

Foster, I., Iamnitchi, A. 2003. *On Death, Taxes, and the Convergence of. Peer-to-Peer and Grid Computing*. IPTPS 2003. LNCS 2735, pp. 118-128. Springer-Verlag Berlin.

Foster, I., 2005, *Globus Toolkit Version 4: Software for Service-Oriented Systems*. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13.

Garey, M., Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY.

Chiang, K., Lloyd, L. 2007. *A case study of the rustock rootkit and spam bot*. In Proceedings of USENIX HotBots'07.

Chord, 2008. *The Chord/DHash Project*. <http://pdos.csail.mit.edu/chord/>. Last accessed 5 April 2009.

Goldsmith, B., 2006, '*CompTorrent: Applying BitTorrent Techniques to Distributed Computing*.' Technical Report, School of Computing, University of Tasmania.

Goldsmith, B., 2007, *Enabling Grassroots Distributed Computing with CompTorrent,* Proceedings of the Sixth International Workshop on Agents and Peer-to-Peer Computing *(AP2PC 2007)*, Honolulu, Hawaii, USA

Goldsmith, B., 2004, *A Peer to Peer Supply Chain Network,* Coursework Masters Thesis, School of Computing, University of Tasmania.

GPU 2007. *GPU: a Global Processing Unit.* <http://gpu.sourceforge.net> Accessed Jan 25, 2007.

Grizzard, J., Sharma, V., Nunnery, C., Kang, B., Dagon, D. 2007. *Peer-to-peer botnets: Overview and case study*. In Proceedings of USENIX HotBots'07.

Gropp, W., Sterling, T., 2003, Beowulf Cluster Computing with Linux, 2nd edition, MIT Press, USA.

Haynes, B., 1998. *Collective Wisdom*. American Scientist, Vol. 86, No. 2, March-April 1998, pp. 118-122.

Hoare, C., 1978. *Communicating sequential processes*. Communications of the ACM 21 (8): pp. 666–677.

Holland, J., 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Horton, M. 1986. *RFC976: UUCP Mail Interchange Format Standard*, Bell Laboratories, USA.

IEEE, 2002. *Computer*. Feburary Issue 2002.

Ingerson, B., Evans, C., Ben-Kiki, O., 2001. *Yet Another Markup Language (YAML) 1.0.* <http://yaml.org/spec/history/2001-08-01.html> Last Accessed 25 Mar 2009.

ICANN 2007. *Factsheet - Root server attack on 6 February 2007*, ICANN (2007-03-01). Last Accessed 15/12/2008.

ITU, 1994. *Open Systems Interconnection - Basic Reference Model: The basic model*, <http://www.itu.int/rec/T-REC-X.200-199407-I/en>, Last Accessed 18/12/2008.

Karbhari, P., Ammar, M., Dhamdhere, A., Raj, H., Riley, G., Zegura, E. 2003. *Bootstrapping in Gnutella: A Preliminary Measurement Study*, Technical Report. Georgia Institute of Technology.

Khan A., Mccreary C., Jones M. 1994. *A comparison of multiprocessor scheduling heuristics*, In Proceedings of the 1994 International Conference on Parallel Processing, volume II, pp. 243-250.

Klensin, J. 2008. *RFC5321: ESMTP*, Simple Mail Transfer Protocol, Network Working Group. <http://tools.ietf.org/html/rfc5321> Last Accessed 25 Mar 2009.

Knuth, D. 1998. *The Art of Computer Programming, Section 5.2.4: Sorting by Merging*, Addison-Wesley. pp.158–168.

Knuth, D. 1998. *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Second Edition. Addison-Wesley. Section 6.3: Digital Searching, pp. 492.

Kotadia M. 2004. SETI's *BOINC project hit by DDoS attack*, <http://news.zdnet.co.uk/security/0,1000000189,39161876,00.htm>

Last Accessed 15/12/2008.

Krauter, K., Buyya, R., Maheswaran , M. 2002, *A taxonomy and survey of grid resource management systems for distributed computing* , Software: Practice and Experience, Vol 32 , No. 2, pp. 135-164 . John Wiley & Sons, Ltd.

Larson, S. M., Snow, D. C., Shirts, M., Pande, V. S. 2003. Folding@Home *and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology*. Grant, R., ed, Horizon Press Modern Methods in Computational Biology.

Leopold, C., 2001, *Parallel and Distributed Computing A survey of models, paradigms and approaches*, pp. 3, Wiley, USA.

Li, Jinyang. 2005, *Routing Tradeoffs in Dynamic Peer-to-peer networks*. PhD Thesis. MIT.

Longbottom, R. 2008. *Whetstone Benchmark History and Results*, <http://freespace.virgin.net/roy.longbottom/whetstone.htm> Last accessed 2 April 2009.

Magoules, F., Nguyen T., Yu L., 2008, *Grid Resource Management*

Towards Virtual And Services Compliant Grid Computing. Taylor & Francis Ltd USA, pp. 98.

McCreary, C., Thompson, J., Gill, D., Smith, T., Zhu, Y. 1993. *Partitioning and Scheduling Using Graph Decomposition*, In Twenty-eighth annual ACM symposium on theory of computing, pp. 93-106.

Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., Venkataramani, A. 2007, *Do Incentives Build Robustness in BitTorrent?* In 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI '07), Cambridge, MA, USA.

Miller, J. 2004, *Characterization of Data on the Gnutella Peer-to-Peer Network*, IEEE Consumer Communications and Networking Conference, January 2004.

Milojicic, D., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z., 2002. *Peer-to-Peer Computing*, HP Laboratories, Technical Report. Palo Alto, HPL-2002-57.

Netperf, 2009, <http://www.netperf.org/>, Last accessed 3 April 2009.

OpenMP, 2009, *The OpenMP API Specification for parallel programming*, <http://openmp.org/>, Last accessed 17 Mar, 2010.

O'Reilly. 2001. The O'Reilly Peer-to-Peer Conference, <http://conferences.oreillynet.com/p2p_sf01/>, Last Accessed 1 Apr 2009.

Plaxton, C., Rajaraman, R., Richa, A. 1997. *Accessing nearby copies of replicated objects in a distributed environment*, ACM Symposium on Parallel Algorithms and Architectures, pp. 311–320.

Postel, J. 1982. *RFC 821: SIMPLE MAIL TRANSFER PROTOCOL*, Information Sciences Institute, University of Southern California.<http://james.apache.org/server/rfclist/smtp/rfc0821.txt>, Last Accessed 3 May 2009.

Protić, J., Tomasevic, M., Milutinović, V. 1997. *Distributed Shared Memory: Concepts and Systems*, Wiley-IEEE Computer Society Press.

Pugh, W. 1990. *Skip lists: A probabilistic alternative to balanced trees*, Communications of the ACM. 33 (6): pp. 668–676.

Radić, B., Imamagić E., 2004. *Benchmarking the Performance of JMS on Computer Clusters*, CARNet Users Conference, 28. 9. 2004.

Ratnasamy, S., Francis, P., Shenker, S., Handley, M., 2001, *A Scalable Content-Addressable Network*, In Proceedings of ACM SIGCOMM , pp. 161 – 172.

Rhea, S., Roscoe, T., Kubiatowicz J. 2003, *Structured Peer-to-peer overlays need application-driven benchmarks*. Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02).

Risson, J., Moors, T. *Survey of Research towards Robust Peer-to-Peer Networks: Search Methods*. 2004. Technical report EE-P2P-1-1, University of New South Wales, Sydney, Australia.

Ritter, J., 2001, *Why Gnutella Can't Scale. No, Really*, <http://www.darkridge.com/~jpr5/doc/gnutella.html>, Last Accessed April 11, 2006.

Rose, M. 2001. *RFC3080: The Blocks Extensible Exchange Protocol Core*, Invisible Worlds, Inc., <http://tools.ietf.org/html/rfc3080>, Last Accessed 4 April 2009.

Rowstron, A., Druschel, P. 2001. *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany. pp. 329–350.

Sedgewick, R. 1990, *Algorithms in C*, 2nd Ed, Addison-Wesley, USA.

Sinnen, O., 2007. *Task Scheduling for Parallel Systems*. Wiley, USA.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J., 1996. *MPI: The Complete Reference.* MIT Press, Boston, USA.

Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H. 2001. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*. In the Proceedings of ACM SIGCOMM 2001, San Deigo, CA.

SYS-CON. 2008, *Twenty-One Experts Define Cloud Computing*, <http://cloudcomputing.sys-con.com/node/612375/print>, Last Accessed April 4, 2009.

Thain, D., Tannenbaum, T., Livny, M., 2005. *Distributed Computing in Practice: the Condor Experience*. Concurrency and Computation: Practice and Experience, Volume 17. pp. 323–356.

Ts, J., Eckstein, R., Collier-Brown, D., 2003. *Using Samba, 2nd Edition*, O'Reilly & Associates, <http://us6.samba.org/samba/docs/using_samba/toc.html>, Last Accessed April 2, 2009.

Vaughan, J., Brookes, G., 1989. *The Mandelbrot set as a parallel processing benchmark*, University Computing, 11, (3), pp. 193–197.

Vixie, P., Sneeringer, G., Schleifer, M. 2002. *Events of 21-Oct-2002*. <http://c.root-servers.org/october21.txt>, Last Accessed 15/12/2008.

Volunteer at Home, 2007, *Volunteer at Home - List of all active volunteer computing projects*.
<http://www.volunteerathome.com/sections/active_projects/active_projects.htm>
Last Accessed 25 Mar 2009.

Wei, B., Fedak, G., Cappello, F., 2005. *Collaborative Data Distribution with BitTorrent for Computational Desktop Grids*, Proceedings of the The 4th International Symposium on Parallel and Distributed Computing (ISPDC'05), p.250-257, July 04-06, 2005.

Weicker, R. *Dhrystone: A Synthetic Systems Programming Benchmark,* Communications of the ACM (CACM), Volume 27, Number 10, October 1984, p. 1013-1030.

Weisstein, E., 2009. *NP-Hard Problem*. From MathWorld--A Wolfram Web Resource. <http://mathworld.wolfram.com/NP-HardProblem.html>. Last accessed 16 May 2009.

Withers, A., 2007. *Condor Scalability and Management at Brookhaven National Laboratory*,
<http://www.cs.wisc.edu/condor/PCW2007/presentations/withers_cw2007.pdf>.
Last accessed 5 April 2009.

Appendix A

The source code for CompTorrent, including the database schema and doxygen documentation (including collaboration diagrams), accompanies the thesis electronically on a DVD.

Building Source Code

These instructions are for building CompTorrent using gcc 4.3.2 on Ubuntu 8.10.

It is assumed that sharutils has been installed on the machine (for uuencode/uudecode).

Obtain the following 3rd party libraries and compile and install them as per their individual instructions.

```
commoncpp2-1.7.3.tar.gz - configure, make, sudo make install
cryptopp560.zip  - make, sudo make install
tinyxml_2_5_3.tar.gz - make
curl-7.19.4.tar.gz - configure, make, sudo make install
```

Installing CompTracker

```
sudo ln -s /home/bcg/source/sp2p/trunk/comptracker/public_html/ comptracker
```

edit globalconfig.php to match your database connection.

```
$global_vars = array(

        "dbhost" => "localhost",
        "dbname" => "sp2p",
        "dbuser" => "root",
        "dbpass" => "77moke",
);
```

Tracker Database Schema

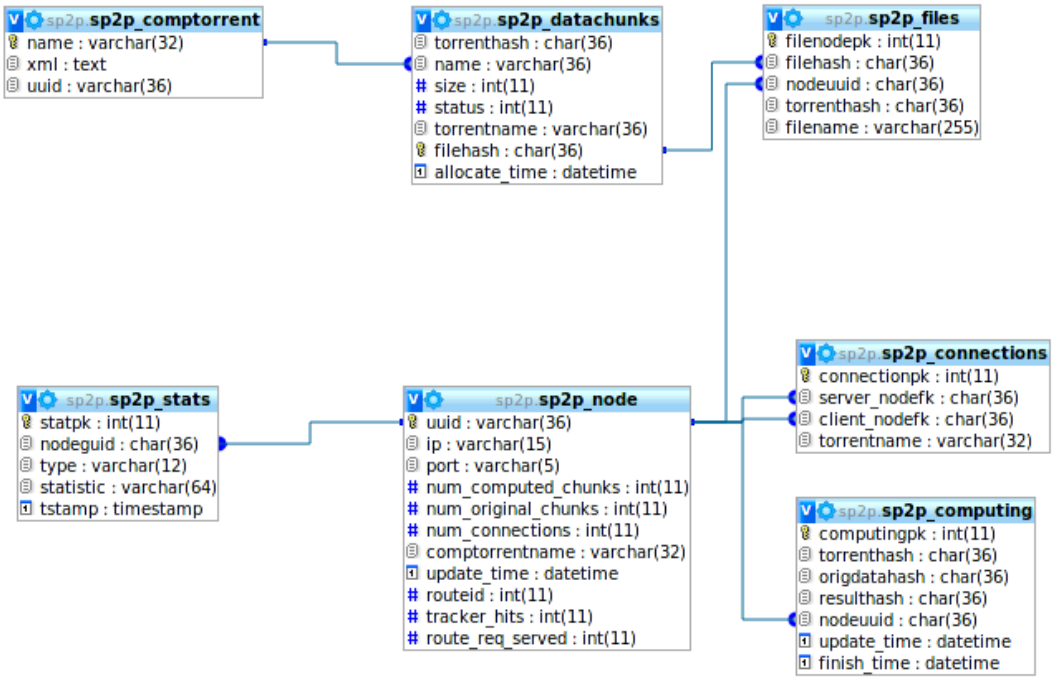
The tracker uses a MySQL database to maintain information about the computing projects its hosting. It provides much of the storage for the tracker interfaces as described in 7.1.2.2.

```
CREATE TABLE `sp2p_comptorrent` (
  `name` varchar(32) NOT NULL,
  `xml` text NOT NULL,
  `uuid` varchar(36) NOT NULL,
  PRIMARY KEY  (`name`),
  KEY `uuid` (`uuid`)
);

CREATE TABLE `sp2p_computing` (
  `computingpk` int(11) NOT NULL auto_increment,
  `torrenthash` char(36) NOT NULL,
```

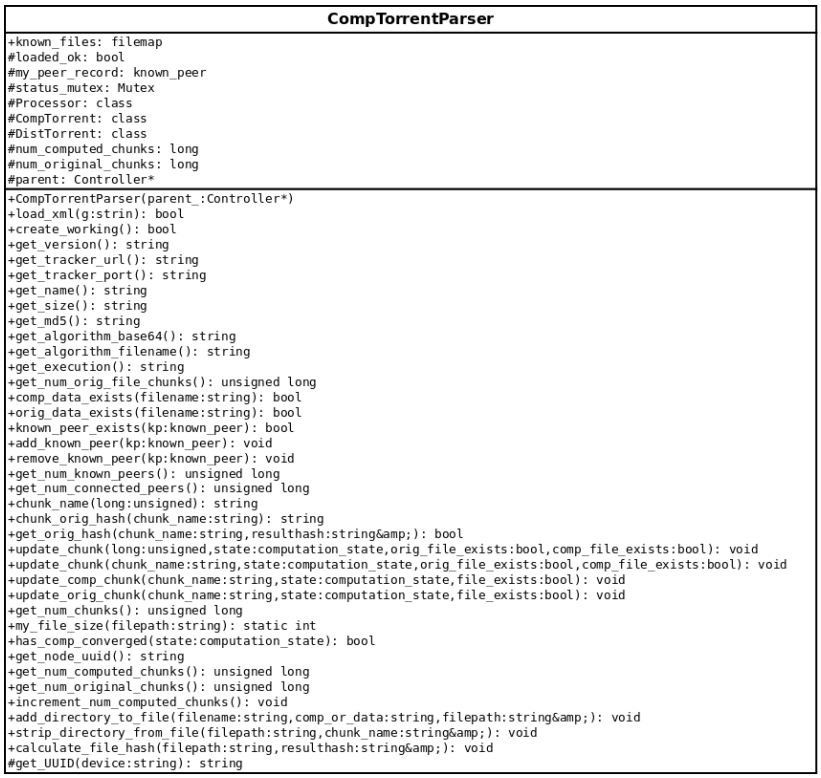
```
  `origdatahash` char(36) NOT NULL,
  `resulthash` char(36) NOT NULL,
  `nodeuuid` char(36) NOT NULL,
  `update_time` datetime NOT NULL,
  `finish_time` datetime NOT NULL,
  `filename` varchar(255) NOT NULL,
  PRIMARY KEY  (`computingpk`),
  KEY `torrenthash` (`torrenthash`),
  KEY `origdatahash` (`origdatahash`,`resulthash`,`nodeuuid`)
);

CREATE TABLE `sp2p_connections` (
  `connectionpk` int(11) NOT NULL auto_increment,
  `server_nodefk` char(36) NOT NULL,
  `client_nodefk` char(36) NOT NULL,
  `torrentname` varchar(32) NOT NULL,
  PRIMARY KEY  (`connectionpk`),
  KEY `server_nodefk` (`server_nodefk`,`client_nodefk`),
  KEY `torrentname` (`torrentname`),
  KEY `client_nodefk` (`client_nodefk`)
);

CREATE TABLE `sp2p_datachunks` (
  `torrenthash` char(36) NOT NULL,
  `name` varchar(36) NOT NULL,
  `size` int(11) NOT NULL,
  `status` int(11) NOT NULL,
  `torrentname` varchar(36) NOT NULL,
  `filehash` char(36) NOT NULL,
  `allocate_time` datetime NOT NULL,
  `num_computed` int(11) NOT NULL,
  PRIMARY KEY  (`name`),
  KEY `torrenthash` (`torrenthash`)
);

CREATE TABLE `sp2p_files` (
  `filenodepk` int(11) NOT NULL auto_increment,
  `filehash` char(36) NOT NULL,
  `nodeuuid` char(36) NOT NULL,
  `torrenthash` char(36) NOT NULL,
  `filename` varchar(255) NOT NULL,
  PRIMARY KEY  (`filenodepk`),
  KEY `filehash` (`filehash`),
  KEY `nodeuuid` (`nodeuuid`),
  KEY `torrenthash` (`torrenthash`)
);

CREATE TABLE `sp2p_ipconnections` (
  `ipconnectionspk` int(11) NOT NULL auto_increment,
  `client` varchar(15) NOT NULL,
  `server` varchar(15) NOT NULL,
  PRIMARY KEY  (`ipconnectionspk`)
);

CREATE TABLE `sp2p_node` (
  `uuid` varchar(36) NOT NULL,
  `ip` varchar(15) NOT NULL,
  `port` varchar(5) NOT NULL,
  `num_computed_chunks` int(11) NOT NULL,
  `num_original_chunks` int(11) NOT NULL,
  `num_connections` int(11) NOT NULL,
  `comptorrentname` varchar(32) NOT NULL,
  `update_time` datetime NOT NULL,
  `routeid` int(11) NOT NULL,
  `tracker_hits` int(11) NOT NULL,
  `route_req_served` int(11) NOT NULL,
  `mutex` varchar(36) NOT NULL,
  PRIMARY KEY  (`uuid`)
);

CREATE TABLE `sp2p_stats` (
  `statpk` int(11) NOT NULL auto_increment,
  `nodeguid` char(36) NOT NULL,
  `type` varchar(12) NOT NULL,
  `statistic` varchar(64) NOT NULL,
  `tstamp` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
```

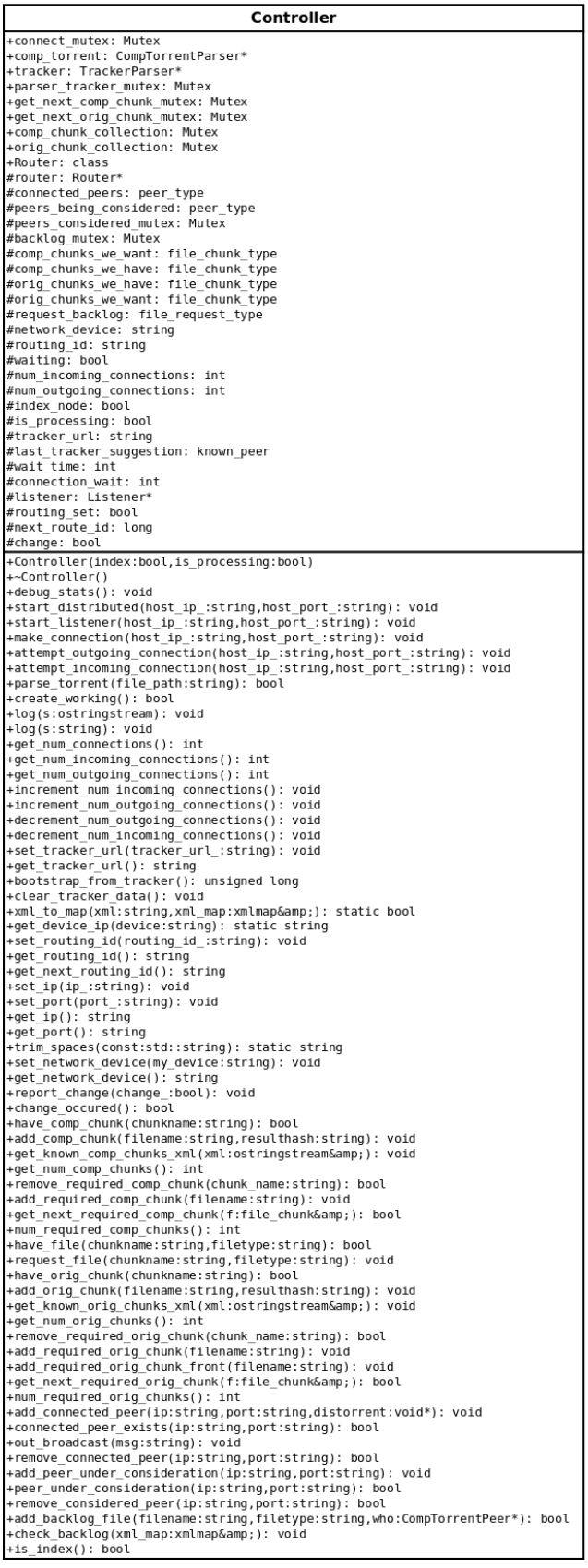
```
  PRIMARY KEY  (`statpk`),
  KEY `nodeguid` (`nodeguid`)
);
```

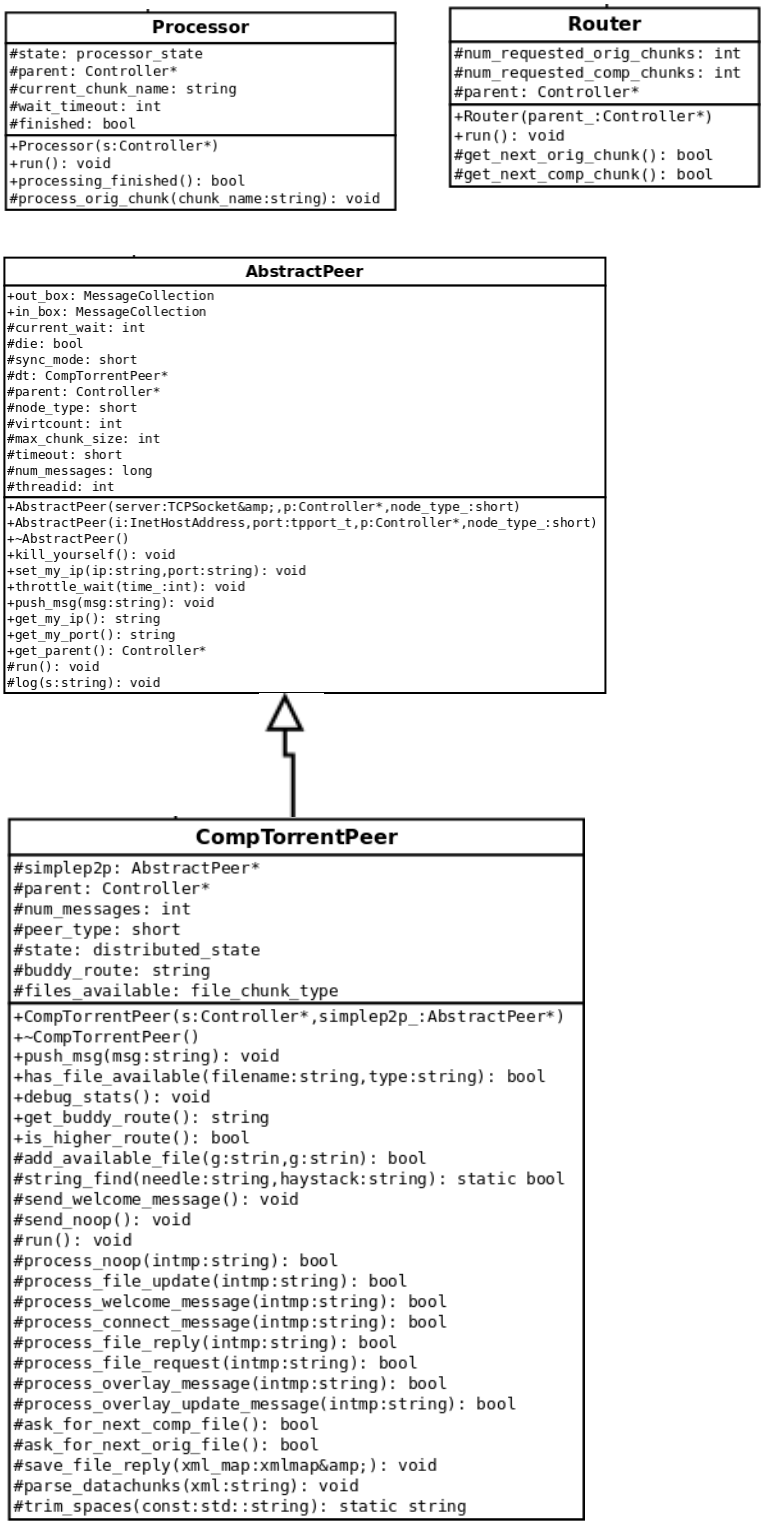
Tracker Database Schema Diagram

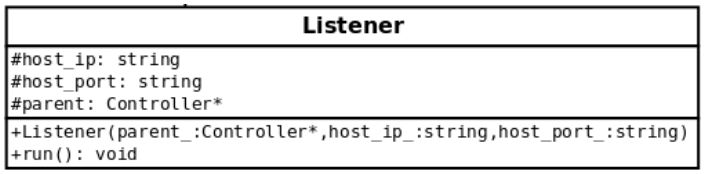


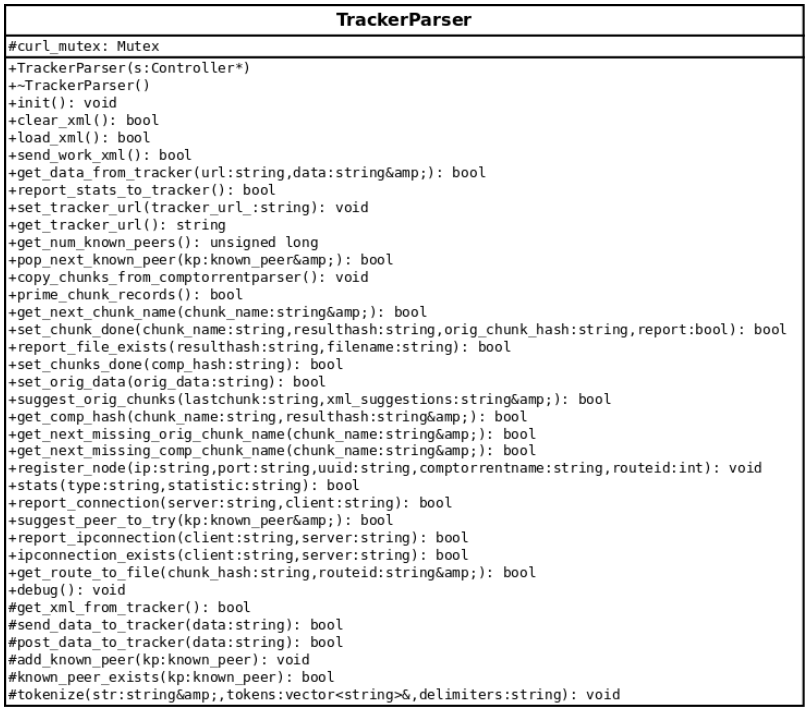
UML Class Diagrams




Controller
+connect_mutex: Mutex
+comp_torrent: CompTorrentParser*
+tracker: TrackerParser*
+parser_tracker_mutex: Mutex
+get_next_comp_chunk_mutex: Mutex
+get_next_orig_chunk_mutex: Mutex
+comp_chunk_collection: Mutex
+orig_chunk_collection: Mutex
+Router: class
#router: Router*
#connected_peers: peer_type
#peers_being_considered: peer_type
#peers_considered_mutex: Mutex
#backlog_mutex: Mutex
#comp_chunks_we_want: file_chunk_type
#comp_chunks_we_have: file_chunk_type
#orig_chunks_we_have: file_chunk_type
#orig_chunks_we_want: file_chunk_type
#request_backlog: file_request_type
#network_device: string
#routing_id: string
#waiting: bool
#num_incoming_connections: int
#num_outgoing_connections: int
#index_node: bool
#is_processing: bool
#tracker_url: string
#last_tracker_suggestion: known_peer
#wait_time: int
#connection_wait: int
#listener: Listener*
#routing_set: bool
#next_route_id: long
#change: bool
+Controller(index:bool,is_processing:bool)
+~Controller()
+debug_stats(): void
+start_distributed(host_ip_:string,host_port_:string): void
+start_listener(host_ip_:string,host_port_:string): void
+make_connection(host_ip_:string,host_port_:string): void
+attempt_outgoing_connection(host_ip_:string,host_port_:string): void
+attempt_incoming_connection(host_ip_:string,host_port_:string): void
+parse_torrent(file_path:string): bool
+create_working(): bool
+log(s:ostringstream): void
+log(s:string): void
+get_num_connections(): int
+get_num_incoming_connections(): int
+get_num_outgoing_connections(): int
+increment_num_incoming_connections(): void
+increment_num_outgoing_connections(): void
+decrement_num_outgoing_connections(): void
+decrement_num_incoming_connections(): void
+set_tracker_url(tracker_url_:string): void
+get_tracker_url(): string
+bootstrap_from_tracker(): unsigned long
+clear_tracker_data(): void
+xml_to_map(xml:string,xml_map:xmlmap&): static bool
+get_device_ip(device:string): static string
+set_routing_id(routing_id_:string): void
+get_routing_id(): string
+get_next_routing_id(): string
+set_ip(ip_:string): void
+set_port(port_:string): void
+get_ip(): string
+get_port(): string
+trim_spaces(const:std::string): static string
+set_network_device(my_device:string): void
+get_network_device(): string
+report_change(change_:bool): void
+change_occured(): bool
+have_comp_chunk(chunkname:string): bool
+add_comp_chunk(filename:string,resulthash:string): void
+get_known_comp_chunks_xml(xml:ostringstream&): void
+get_num_comp_chunks(): int
+remove_required_comp_chunk(chunk_name:string): bool
+add_required_comp_chunk(filename:string): void
+get_next_required_comp_chunk(f:file_chunk&): bool
+num_required_comp_chunks(): int
+have_file(chunkname:string,filetype:string): bool
+request_file(chunkname:string,filetype:string): void
+have_orig_chunk(chunkname:string): bool
+add_orig_chunk(filename:string,resulthash:string): void
+get_known_orig_chunks_xml(xml:ostringstream&): void
+get_num_orig_chunks(): int
+remove_required_orig_chunk(chunk_name:string): bool
+add_required_orig_chunk(filename:string): void
+add_required_orig_chunk_front(filename:string): void
+get_next_required_orig_chunk(f:file_chunk&): bool
+num_required_orig_chunks(): int
+add_connected_peer(ip:string,port:string,distorrent:void*): void
+connected_peer_exists(ip:string,port:string): bool
+out_broadcast(msg:string): void
+remove_connected_peer(ip:string,port:string): bool
+add_peer_under_consideration(ip:string,port:string): void
+peer_under_consideration(ip:string,port:string): bool
+remove_considered_peer(ip:string,port:string): bool
+add_backlog_file(filename:string,filetype:string,who:CompTorrentPeer*): bool
+check_backlog(xml_map:xmlmap&): void
+is_index(): bool


Processor
#state: processor_state
#parent: Controller*
#current_chunk_name: string
#wait_timeout: int
#finished: bool
+Processor(s:Controller*)
+run(): void
+processing_finished(): bool
#process_orig_chunk(chunk_name:string): void
Router
#num_requested_orig_chunks: int
#num_requested_comp_chunks: int
#parent: Controller*
+Router(parent_:Controller*)
+run(): void
#get_next_orig_chunk(): bool
#get_next_comp_chunk(): bool
AbstractPeer
+out_box: MessageCollection
+in_box: MessageCollection
#current_wait: int
#die: bool
#sync_mode: short
#dt: CompTorrentPeer*
#parent: Controller*
#node_type: short
#virtcount: int
#max_chunk_size: int
#timeout: short
#num_messages: long
#threadid: int
+AbstractPeer(server:TCPSocket&,p:Controller*,node_type_:short)
+AbstractPeer(i:InetHostAddress,port:tpport_t,p:Controller*,node_type_:short)
+~AbstractPeer()
+kill_yourself(): void
+set_my_ip(ip:string,port:string): void
+throttle_wait(time_:int): void
+push_msg(msg:string): void
+get_my_ip(): string
+get_my_port(): string
+get_parent(): Controller*
#run(): void
#log(s:string): void
CompTorrentPeer
#simplep2p: AbstractPeer*
#parent: Controller*
#num_messages: int
#peer_type: short
#state: distributed_state
#buddy_route: string
#files_available: file_chunk_type
+CompTorrentPeer(s:Controller*,simplep2p_:AbstractPeer*)
+~CompTorrentPeer()
+push_msg(msg:string): void
+has_file_available(filename:string,type:string): bool
+debug_stats(): void
+get_buddy_route(): string
+is_higher_route(): bool
#add_available_file(g:strin,g:strin): bool
#string_find(needle:string,haystack:string): static bool
#send_welcome_message(): void
#send_noop(): void
#run(): void
#process_noop(intmp:string): bool
#process_file_update(intmp:string): bool
#process_welcome_message(intmp:string): bool
#process_connect_message(intmp:string): bool
#process_file_reply(intmp:string): bool
#process_file_request(intmp:string): bool
#process_overlay_message(intmp:string): bool
#process_overlay_update_message(intmp:string): bool
#ask_for_next_comp_file(): bool
#ask_for_next_orig_file(): bool
#save_file_reply(xml_map:xmlmap&): void
#parse_datachunks(xml:string): void
#trim_spaces(const:std::string): static string


Listener
#host_ip: string
#host_port: string
#parent: Controller*
+Listener(parent_:Controller*,host_ip_:string,host_port_:string)
+run(): void


TrackerParser
#curl_mutex: Mutex
+TrackerParser(s:Controller*)
+~TrackerParser()
+init(): void
+clear_xml(): bool
+load_xml(): bool
+send_work_xml(): bool
+get_data_from_tracker(url:string,data:string&): bool
+report_stats_to_tracker(): bool
+set_tracker_url(tracker_url_:string): void
+get_tracker_url(): string
+get_num_known_peers(): unsigned long
+pop_next_known_peer(kp:known_peer&): bool
+copy_chunks_from_comptorrentparser(): void
+prime_chunk_records(): bool
+get_next_chunk_name(chunk_name:string&): bool
+set_chunk_done(chunk_name:string,resulthash:string,orig_chunk_hash:string,report:bool): bool
+report_file_exists(resulthash:string,filename:string): bool
+set_chunks_done(comp_hash:string): bool
+set_orig_data(orig_data:string): bool
+suggest_orig_chunks(lastchunk:string,xml_suggestions:string&): bool
+get_comp_hash(chunk_name:string,resulthash:string&): bool
+get_next_missing_orig_chunk_name(chunk_name:string&): bool
+get_next_missing_comp_chunk_name(chunk_name:string&): bool
+register_node(ip:string,port:string,uuid:string,comptorrentname:string,routeid:int): void
+stats(type:string,statistic:string): bool
+report_connection(server:string,client:string): bool
+suggest_peer_to_try(kp:known_peer&): bool
+report_ipconnection(client:string,server:string): bool
+ipconnection_exists(client:string,server:string): bool
+get_route_to_file(chunk_hash:string,routeid:string&): bool
+debug(): void
#get_xml_from_tracker(): bool
#send_data_to_tracker(data:string): bool
#post_data_to_tracker(data:string): bool
#add_known_peer(kp:known_peer): void
#known_peer_exists(kp:known_peer): bool
#tokenize(str:string&,tokens:vector<string>&,delimiters:string): void

Appendix B

Amdahl's Law

Amdahl's Law (Amdahl 1967) explains that all programs have a limit to which they can be parallelised, and gives us an algorithm with which to calculate the maximum possible speedup of a parallel program. The limitation arises from the non parallelisable part of the application, the part that must be executed sequentially.

$$\mathbf{S} = \frac{1}{(1-P) + \frac{P}{N}}$$

Where:

- S is the speedup of the program,.
- P is the part of the application that can be parallelised,
- (1 − P) is the part of the program which must be executed sequentially.
- N is the number of processing elements working on the task.

As N grows large the maximum speedup tends towards (1 − P). This gives an upper limit the amount of processors can be usefully utilised in a parallel computing exercise for a particular application or algorithm.

Gustafson-Barsis' Law

Gustafson-Barsis' Law allows for more flexibility over Amdahl's Law by allowing for the number of processors to change and the size of the problem to change as well during computation. This makes Gustafson-Barsis' Law much more suitable for modelling volunteer networks where nodes come and go and the overall problem size is often in flux.

$$\mathbf{S(P) = P - \alpha\ (P - 1)}$$

Where:

- S is the speedup of the application.
- P is the number of processing elements.
- α is the part of the algorithm that is sequential.

Gustafson-Barsis' Law achieves this through considering the overall time of sequential execution rather than a fixed amount of sequential execution per processor.

Appendix C

Experimental Setup Instructions

This thesis contains experiments covering several different existing distributed computing systems. These systems are non-trivial to install, configure and operate. This appendix covers the installation instructions and settings used in detail for the experimental results presented in this thesis. These should serve as a starting point for building on or verification of this work for BOINC and Condor (See section 6.3 for instructions for CompTorrent).

In these examples the address 144.6.40.251 and 127.0.0.1 are used interchangeably to represent the master server machine. The user name "bcg" is used to represent the logged in account of the server administrator.

BOINC

The host system was Ubuntu Linux 6.06 with a full complement of GNU development tools, mysql and php installed via apt-get.

Server

Check out the code thus:

```
svn co http://boinc.berkeley.edu/svn/trunk/boinc
```

(Revision 14015 was the latest revision at time of experimentation).

Build the code:

```
./_autosetup
./configure --disable-client
make
```

Create a project:

```
./make_project --url_base http://144.6.40.251/boinc/ --db_host localhost
--db_user root test1
```

Steps to complete installation:

1. Change Apache configuration (as root):

```
cat /home/bcg/projects/test1/test1.httpd.conf >> /etc/apache/httpd.conf && apachectl
restart
```

(note: path to httpd.conf varies)

```
sudo cp /home/bcg/projects/test1/test1.httpd.conf /etc/apache2/sites-available/
```

2. Add to crontab (as bcg (my username) in this case)

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * /home/bcg/projects/test1/bin/start --cron
```

(If cron cannot run "start", try using a helper script to set PATH and PYTHONPATH)

3. The project is configured with a test application.

To install this application (recommended) run:

```
cd /home/bcg/projects/test1
bin/xadd
bin/update_versions
```

To start, show status, and stop the project, run:

```
bin/start
bin/status
bin/stop
```

The project's URLs are (the url and project name will of course depend on installation):

Home page (and master URL): http://144.6.40.251/test1/

Administrative page:http://144.6.40.251/test1_ops/

Enable account creation – edit config.xml in the ~/project/test1/ directory and set the the disable_account_creation element to 0

BOINC connects to Internet URLs in a few places in the code. This poses an annoying problem behind university proxy servers. To alleviate this, comment out the following lines of code:

/home/bcg/projects/test1/html/ops/index.php

Insert a /* at line 23 and a */ at the end of line 39.

/home/bcg/projects/test1/html/inc/user.inc

Modify the function get_other_projects at line 60 to return $user and do nothing else eg:

```
function get_other_projects($user) {
        /* $cpid = md5($user->cross_project_id . $user->email_addr);
        $url = "http://boinc.netsoft-online.com/get_user.php?cpid=$cpid";
        $f = fopen($url, "r");
        if (!$f) {
                return $user;
        }
        $u = parse_user($f, $user);
        fclose($f);
        return $u; */
        return $user;
}
```

Client Side

Download boinc_5.10.21_i686-pc-linux-gnu.sh

```
./boinc --attach_project http://144.6.40.251/test1/
de68da8503386b0232e664a27f2dad92
```

That should be all that is required however here are some problems encountered and solutions:

Problem: Getting the authenticator for each client.

Solution: Get it from the db.

Problem: No work from project. The feeder seemed to be dying as can be seen by the status.

bcg@rhdl-a2:~/projects/test1/bin$ ./status

```
BOINC is ENABLED
DAEMON pid status   lockfile disabled commandline
 1  7907 NOT RUNNING UNLOCKED no    feeder -d 3
 2  7909 running   locked  no    transitioner -d 3
 3  7912 running   locked  no    file_deleter -d 3
```

Tried running the feeder directly (as it would be by BOINC as defined in config.xml):

```
bcg@rhdl-a2:~/projects/test1/bin$ ./feeder -d 3
2007-11-20 16:42:29.7621 [normal ] Starting
shmctl RMID: Operation not permitted
2007-11-20 16:42:29.7631 [CRITICAL] can't destroy shmem
```

Solution:

change the group for the feeder thus:

```
bcg@rhdl-a2:~/projects/test1/bin$ sudo chgrp www-data feeder
bcg@rhdl-a2:~/projects/test1/bin$ sudo chmod g+xs feeder
```

But if that doesn't work, execute the start with sudo:

```
sudo bin/start
```

and that should be problem solved.

Here is an example for setting up a project using Mandelbrot:

Server:

Create the Project

```
cd /data/source/boinc/tools/
./make_project --url_base http://144.6.40.251/ --db_host localhost
--db_user root mandelbrot16
```

Set up the Project Website

```
sudo cp mandelbrot16.httpd.conf /etc/apache2/sites-available/

sudo ln -s /etc/apache2/sites-available/mandelbrot16.httpd.conf
/etc/apache2/sites-enabled/mandelbrot16.httpd.conf

sudo apache2ctl restart

cd ~/projects/mandelbrot16

sudo chown www-data ~/projects/ -R
```

Enable Account Creation

Enable account creation – edit config.xml in the ~/project/mandelbrot16/ directory and set the the disable_account_creation element to 0

Edit the Project

In this example, only Linux machines are being used, so par down the project xml to only include one target platform (or adjust for your circumstances).

Also, add in the name of the algorithm executable. In this case its Mandelbrot.

```
<boinc>
  <platform>
    <name>i686-pc-linux-gnu</name>
    <user_friendly_name>Linux running on an Intel x86-compatible
CPU</user_friendly_name>
  </platform>
  <app>
    <name>mandelbrot</name>
    <user_friendly_name>mandelbrot16</user_friendly_name>
  </app>
</boinc>
```

Edit config.xml

Make sure the db_passwd element contains your db password

Compile the Wrapper

```
svn co http://boinc.berkeley.edu/svn/trunk/boinc_samples
cd boinc_samples/wrapper
ln -s `g++ -print-file-name=libstdc++.a`
make
cp boinc_samplaes/wrapper/wrapper
~/projects/mandelbrot16/apps/mandelbrot/wrapper_5.5_i686-pc-linux-
gnu
```

Copy the Algorithm

Also make sure to name your algorithm as per BOINC's naming convention so that it matches the target platforms set out in the project.xml.

```
mkdir ~/projects/mandelbrot16/apps/mandelbrot/
cp
/data/source/sp2p/experiments/mandelbrot/160/comptorrents/working/mandelbrot/mandelbr
ot ~/projects/mandelbrot16/apps/mandelbrot_5.5_i686-pc-linux-gnu
bin/xadd
bin/update_versions
```

Add Templates for Work Units and Results

Put the following into a file called result_template in ~/projects/mandelbrot16/templates/

```
  <file_info>
    <name><OUTFILE_0/></name>
```

```
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>1000000</max_nbytes>
    <url><UPLOAD_URL/></url>
  </file_info>
  <result>
    <file_ref>
      <file_name><OUTFILE_0/></file_name>
      <open_name>out</open_name>
    </file_ref>
  </result>
```

Put the following into a file called work_unit_template in ~/projects/mandelbrot16/templates/

```
<file_info>
 <number>0</number>
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>in</open_name>
  </file_ref>
</workunit>
```

Add Work Units

The mandelbrot executable takes 2 command line arguments: an infile and an outfile.

The infile stipulates the parameters for the mandelbrot set being generated.

The outfile is a resulting jpeg.

For this example, there are 16 work units, each a file containing the settings for the overall Mandelbrot set as well as the region of the set to be calculated (so it can be split up and computed over multiple machines).

Copy each of these files into the ~/projects/mandelbrot16/download directory.

Now, tell BOINC about each one of these files by inserting each one of them as a work unit (using the templates from the previous step).

```
bin/create_work -appname mandelbrot -wu_name mandelbrot_00000001 -wu_template
templates/work_unit_template -result_template templates/result_template -min_quorum 1
-target_nresults 1 mandelbrot_00000001
```

Do this changing the file and work unit name for each data set file eg. mandelbrot_00000002, mandelbrot_00000003 and so on.

Start BOINC

```
sudo bin/start
```

You can also stop it (sudo bin/stop) and get a status (sudo bin/status)

Create a Client User Account

Go to the project homepage (http://144.6.40.251/mandelbrot16/) and create an account.

If the server does not have public Internet access, as is often the case on a dedicated cluster, this page might take a while to load as it tries to get user info from the BOINC main site. You can hack this out editing the file user.inc (home/bcg/projects/test1/html/inc/user.inc). Modify the function get_other_projects at line 60 to return $user and do nothing else eg:

```
function get_other_projects($user) {
      /* $cpid = md5($user->cross_project_id . $user->email_addr);
      $url = "http://boinc.netsoft-online.com/get_user.php?cpid=$cpid";
      $f = fopen($url, "r");
      if (!$f) {
            return $user;
      }
      $u = parse_user($f, $user);
      fclose($f);
      return $u; */
      return $user;
}
```

Since this is a dedicated cluster example, you like me will probably not bother setting up email on the servers. You need to edit the created user's database entry and also get the authentication key so command line clients can attach to the project.

Connect to the database using phpmyadmin or similar. Go to the mandelbrot16 database and browse the results in the user table.

Take note of the authenticator field value (eg. 84a35ba7615192bd3120019d8861ffac). You will need to to connect shortly.

Update the email_validated field to contain a 1.

Client Machines:

Download boinc_5.10.21_i686-pc-linux-gnu.sh from the BOINC website.

Run it on the client machine from your home directory. This will install the runtime files into a BOINC directory.

In a terminal run:

```
./run_client
```

Alternatively, you can create the following script (from: http://blog.os-tools.net/?p=31):

```
#!/bin/sh
# /etc/init.d/boinc
# Start/stop/restart

test -x /home/bcg/BOINC/boinc || exit 0

case "$1" in
start)
echo "Starting BOINC."
cd "/home/bcg/BOINC" && exec ./boinc >>stdoutdae.txt
2>>stderrdae.txt &
;;
stop)
echo "Stopping BOINC."
killall boinc
;;
restart)
killproc boinc
sleep 2
cd "/home/bcg/BOINC" && exec ./boinc >>stdoutdae.txt
2>>stderrdae.txt &
;;
*)
echo "Usage: /etc/init.d/boinc {start|stop|restart}"
exit 2
esac
exit 0
```

Which will allow you to start and stop BOINC as a service – so you can maintain one terminal window only in which to run the client and then issue commands to it. This saves having 2 x n terminals open (where n = number of nodes). 16 cshh windows is bad enough without needing 32 open on a single desktop.

Attach to the Project

In another separate terminal window:

```
./boinc_cmd –project_attach http://144.6.40.251/mandelbrot16/mandelbrot16/
cc5e5948e2ff9fe00dc2474d271753ad
```

Use your own authenticator at this point that you noted down earlier.

You can also detach from the project later on by running:

```
./boinc_cmd –project http://144.6.40.251/mandelbrot16/ detach
```

To later reset projects:

Stop BOINC server.
Empty the workunit and result tables in the database.
empty the ~projects/mandelbrot16/upload directory
Add work units again.
Start BOINC server.

Condor

Master Machine

Download condor-6.8.7-linux-x86-rhel.tar.gz and uncompress it.

```
cd /data/condor-6.8.7

mkdir condor_root
mkdir condor_local

sudo ./condor_configure --install-dir=/data/condor-6.8.7/condor_root/
--type=manager,submit --local-dir=/data/condor-6.8.7/condor_local/ --owner=bcg
--install=/data/condor-6.8.7/release.tar

export CONDOR_CONFIG=/data/condor-6.8.7/condor_root/etc/condor_config
```

Edit condor_root/etc/condor_config:

Set RELEASE_DIR to /data/condor-6.8.7/condor_root/

Set HOSTALLOW_WRITE to *

Set HOSTALLOW_ADMINISTRATOR = $(FULL_HOSTNAME)

Start Condor:

sudo condor_root/sbin/condor_master

Stop Condor:

sudo condor_root/sbin/condor_off -master

Check that its running:

ps -ef | egrep condor_

```
bcg@rhdl-a2:/data/condor-6.8.7$ ps -ef | egrep condor_
bcg      24421     1  0 12:25 ?        00:00:00 condor_root/sbin/condor_master
bcg      24422 24421  0 12:25 ?        00:00:00 condor_collector -f
bcg      24423 24421  0 12:25 ?        00:00:00 condor_negotiator -f
```

```
bcg      24424 24421  0 12:25 ?        00:00:00 condor_schedd -f
bcg      24425 24421  7 12:25 ?        00:00:07 condor_startd -f
bcg      24475  5431  0 12:27 pts/0    00:00:00 grep -E condor_
```

Create a job

```
# file name:  mandelbrot16.condor
# Condor submit description file for mandelbrot
Executable      = path_to/mandelbrot
Universe        = vanilla
Error           = logs/err.$(cluster)
Output          = logs/out.$(cluster)
Log             = logs/log.$(cluster)

should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = files/mandelbrot_00000001
Arguments       = mandelbrot_00000001 mandelbrot_00000001_out
Queue
```

create a directory to put the job in.

```
mkdir mandelbrot16
cd mandelbrot16
mkdir logs
mkdir files
```

Submit a job

```
condor_root/bin/condor_submit mandelbrot16/mandelbrot16.condor
```

Check on jobs

All jobs:

```
bin/condor_q
```

A job:

```
bin/condor_q 3
```

Client Machine

Download condor-6.8.7-linux-x86-rhel.tar.gz and uncompress it.

```
cd /data/condor-6.8.7
mkdir condor_root
mkdir condor_local
sudo ./condor_configure --install-dir=/home/bcg/condor-6.8.7/condor_root/
--type=execute --local-dir=/home/bcg/condor-6.8.7/condor_local/ --owner=bcg
--install=/home/bcg/condor-6.8.7/release.tar
export CONDOR_CONFIG=/home/bcg/condor-6.8.7/condor_root/etc/condor_config
```

Edit condor-6.8.7/condor_root/etc/condor_config

```
Set UID_DOMAIN = $(FULL_HOSTNAME)
Set FILESYSTEM_DOMAIN=$(FULL_HOSTNAME)
Set HOSTALLOW_ADMINISTRATOR = $(FULL_HOSTNAME)
```

```
Set HOSTALLOW_WRITE to *
```

Edit condor-6.8.7/condor_local/condor_config.local

```
Set CONDOR_HOST = 144.6.40.251
SET UID_DOMAIN and FILESYSTEM_DOMAIN to $(FULL_HOSTNAME)

NETWORK_INTERFACE = 144.6.40.115
WANT_SUSPEND = FALSE
CONTINUE = TRUE
SUSPEND = FALSE
PREEMPT = FALSE
START=TRUE
```

Appendix D

WAN-DC Extras

Sequential read/write with underlying network changes (synthetic)

This is a variation on the earlier Sequential read/write test with the addition of modulating the performance of the underlying network.

Test Description	Perform the Sequential read/write test whilst changing the bandwidth and latency properties of the underlying network.
Test Aim	To observe how the system performs distributing test and result data under a variety of network conditions.
Input Data	A freely, Internet-available dataset of a suitable size for the test. Suggested sizes ranging from 100kb, 1000kb, 10Mb, 100Mb, 1Gb. The Internet Archive is a suggested repository for source data.
Output Data	Wall clock time to complete the file distribution from start to finish. Mbytes/sec if the tested system provides its own figure.
Method	1. Homogeneous Modulate the bandwidth of the underlying network to represent commonly available device bandwidths ranging from modem speeds to gigabit Ethernet. All devices should be set to the same speed. Apply the method of the sequential read/write for both uniform size and mixed data unit sizes. 2. Heterogeneous Modulate the bandwidth of the underlying network to represent commonly available device bandwidths ranging from modem speeds to gigabit Ethernet. All devices should be set to a range of different speeds to represent a known mix of devices. Apply the method of the sequential read/write for both uniform size and mixed data unit sizes.

Input/Output Intensive

The benchmark presented here is designed to compare the ability of cluster technologies to distributed data between nodes on the network.

Sequential read/write (synthetic / application)

The aim of this benchmark is to test how quickly a distributed computing system can copy data between nodes. For systems that use NFS (or similar) as their file system, this will ostensibly be a test of that technology, the network connection and the cluster hardware. For systems that divide data into discrete units and share it as packets, this will enable the measurement of the packet protocol, the network connection and the cluster hardware.

Test Description	Obtain bandwidth figures (Mbytes/sec) and wall clock time to distribute data.
Test Aim	To gain an understanding of the efficiency of the system to distribute test and result data across the network using a variety of data set sizes.
Input Data	A freely, Internet-available dataset of a suitable size for the test. Suggested sizes ranging from 100kb, 1000kb, 10Mb, 100Mb, 1Gb. The Internet Archive is a suggested repository for source data.
Output Data	Wall clock time to complete the file distribution from start to finish. Mbytes/sec if the tested system provides its own figure.
Method	1. Uniform size For each data set size, each set should be split into smaller uniform elements to simulate it being a unit of work for processing. Suggested sizes ranging from 512bytes, 1kb, 10kb, 100kb, 256kb, 1000kb, 10Mb, 20Mb, 50Mb, 100Mb. 2. Mix For each data set size, each set should be split into smaller random sized elements to simulate it being a unit of work for processing. Suggested sizes including 512bytes, 1kb, 10kb, 100kb, 256kb,1000kb, 10Mb, 20Mb, 50Mb, 100Mb.

BitTorrent (theoretical maximum)

Test Description	Obtain bandwidth figures (Mbytes/sec) and wall clock time to distribute data over the cluster using the BitTorrent protocol.
Test Aim	This test is an attempt to gain a best estimate of maximum disk and network performance of the cluster hardware. This is to serve as a working theoretical maximum for what a distributed computing system could hope to achieve.
Input Data	A freely, Internet-available dataset of a suitable size for the test. Suggested sizes ranging from 100kb, 1000kb, 10Mb, 100Mb, 1Gb. The Internet Archive is a suggested repository for source data.
Output Data	BitTorrent timing output for each worker node in the cluster (produced by application with results to be tabulated by tester). Wall clock time to complete the file distribution from start to finish.

Method	Application should be run from 1 worker to all worker machines with a reasonable number of steps to meet a stated confidence interval (95% suggested). Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin. Care should be taken to ensure that one of the tested message chunk sizes corresponds to one of the sizes tried in sequential read/write. BitTorrent's usual size is 256kb.

Test Description	Perform the Mandelbrot benchmark under varying degrees of network load.
Test Aim	Load the network with traffic to investigate the effect of heavy traffic up to a denial of service attack on various components of the distributed computing system.
Input Data	Nil
Output Data	Wall clock time to complete the Mandelbrot set.
Method	1. General load Load the network with varying degrees of traffic using a tool such as WebLOAD. 2. Denial of Service Load individual parts of the distributed computing system with varying degrees of traffic. Individual clients, server components, name servers and other components, as per the system being tested, should be loaded in turn to examine the effect it has on performance.

Failure and Malfeasance

Distributed computing systems that operate over wide area networks must consider the uncertain nature of nodes providing services to the system.

Erroneous Results (synthetic)

Incorrect results

Test Description	Perform the Mandelbrot benchmark with deliberately introduced errors at a known probabilistic rate.
Test Aim	To understand how the system copes with varying degrees of incorrect results and how node exclusion policies can effect the overall process.
Input Data	A set of files each representing the region of the Mandelbrot set to calculate. There are three separate parameter sets to this bench mark. 1. The overall set size being generated. 2. The number of segments the set is to be split into to be computed in parallel. 3. The region of the Mandelbrot set to be calculated. 4. The probability of this work unit being calculated incorrectly. Code has been provided to randomly change some points within the calculated work unit at a particular probabilistic rate. Start with a low probability, i.e. 0.0001, and then step up towards 1 in reasonable increments.

	Depending on items 2 and 3, each work unit can be either relatively uniform or provide a mix ranging from intensive computation to near trivial. Two datasets are provided, Mandelbrot_A and Mandelbrot_B, which illustrate each option.
Output Data	Image files of the Mandelbrot set. The wall clock time to produce the set.
Method	Perform this test using one of the same sets of parameters as the standard Mandelbrot to give a starting point. 1. Random node failure Arrange the work units so that all nodes have the same probability of failure. 2. Bad node Select a single node as random to be a source of error at a probabilistic rate.

First work unit (synthetic derived from application)

Clock time taken for a work unit to be completed when measured from the start of computation.

Test Description	Obtain wall clock times for a first work unit to be received.
Test Aim	To investigate the lag, setup or joining time that occurs in a distributed computing system. This is especially important for comparing emerging systems that rely on a decentralised approach where a bootstrapping phase may occur as a node joins the network.
Input Data	Data can be gathered from the results of other benchmark experiments from both a server or node perspective. That is, the time taken for the first completed work unit to be received. Or, the time between starting a new worker node and actually receiving data to process (where data is available).
Output Data	The elapsed time in microseconds.
Method	Application should be run from 1 worker to all worker machines with a reasonable number of steps to meet a stated confidence interval (95% suggested) for each work unit length. Each step should be repeated multiple times in order to gauge variability. Statistics provided should carefully describe the final values given in terms of their origin. Care should be taken to ensure that the test is fair and that all systems being compared are set to begin computation as soon as possible and that network load is comparable between tests.

Appendix E

Data Sets

Transcode

The file CC_1914_03_26_CruelCruelLove.mpg is available from archive.org and has been split using the mpgtx tool thus:

```
mpgtx -200 ../CC_1914_03_26_CruelCruelLove.mpg -b CC_1914_03_26_CruelCruelLove
```

The individual files as used here are also available with this thesis and listed below with their sizes in bytes.

1035738 chunk-001.mpg	899402 chunk-035.mpg	992574 chunk-068.mpg
961558 chunk-002.mpg	996380 chunk-036.mpg	998936 chunk-069.mpg
997946 chunk-003.mpg	992284 chunk-037.mpg	741322 chunk-070.mpg
836572 chunk-004.mpg	749924 chunk-038.mpg	995420 chunk-071.mpg
995420 chunk-005.mpg	933264 chunk-039.mpg	993564 chunk-072.mpg
995420 chunk-006.mpg	1024964 chunk-040.mpg	980840 chunk-073.mpg
747462 chunk-007.mpg	1006002 chunk-041.mpg	813808 chunk-074.mpg
993884 chunk-008.mpg	746822 chunk-042.mpg	998552 chunk-075.mpg
995834 chunk-009.mpg	989344 chunk-043.mpg	989088 chunk-076.mpg
997626 chunk-010.mpg	1000408 chunk-044.mpg	748550 chunk-077.mpg
858346 chunk-011.mpg	799612 chunk-045.mpg	998680 chunk-078.mpg
999768 chunk-012.mpg	975434 chunk-046.mpg	993564 chunk-079.mpg
747048 chunk-013.mpg	1020616 chunk-047.mpg	997020 chunk-080.mpg
991900 chunk-014.mpg	991934 chunk-048.mpg	748388 chunk-081.mpg
999192 chunk-015.mpg	744616 chunk-049.mpg	991486 chunk-082.mpg
747304 chunk-016.mpg	994524 chunk-050.mpg	999704 chunk-083.mpg
998424 chunk-017.mpg	998488 chunk-051.mpg	1025762 chunk-084.mpg
993756 chunk-018.mpg	995804 chunk-052.mpg	745512 chunk-085.mpg
998202 chunk-019.mpg	743720 chunk-053.mpg	993692 chunk-086.mpg
749156 chunk-020.mpg	998138 chunk-054.mpg	995676 chunk-087.mpg

993884 chunk-021.mpg	976458 chunk-055.mpg	746344 chunk-088.mpg
1004086 chunk-022.mpg	798080 chunk-056.mpg	1146148 chunk-089.mpg
812272 chunk-023.mpg	1000792 chunk-057.mpg	795264 chunk-090.mpg
988256 chunk-024.mpg	997626 chunk-058.mpg	998872 chunk-091.mpg
997370 chunk-025.mpg	922678 chunk-059.mpg	995420 chunk-092.mpg
996666 chunk-026.mpg	799100 chunk-060.mpg	854894 chunk-093.mpg
812310 chunk-027.mpg	930640 chunk-061.mpg	904932 chunk-094.mpg
980042 chunk-028.mpg	997562 chunk-062.mpg	996986 chunk-095.mpg
995706 chunk-029.mpg	994938 chunk-063.mpg	995676 chunk-096.mpg
841082 chunk-030.mpg	916350 chunk-064.mpg	937322 chunk-097.mpg
1000920 chunk-031.mpg	992796 chunk-065.mpg	830658 chunk-098.mpg
747876 chunk-032.mpg	747014 chunk-066.mpg	1016042 chunk-099.mpg
997178 chunk-033.mpg	997690 chunk-067.mpg	703680 chunk-100.mpg
996218 chunk-034.mpg		

POV-Ray

The files benchmark.pov and benchmark.ini are widely available and accompany this thesis.

Mandelbrot

A typical configuration of the Mandelbrot set is rendered in vertical slices of 40 pixels wide with an overall images size of 6400 by 4800.

Mandelbrot_00000001:

```
-2.3333166666666667 1.00001666666666666 -1.25 1.25 0 0 40 4800 1024 6400 4800
```

mandelbrot_00000002 :

```
-2.3333166666666667 1.00001666666666666 -1.25 1.25 40 0 80 4800 1024
6400 4800
```

...

mandelbrot_00000159 :

```
-2.3333166666666667 1.00001666666666666 -1.25 1.25 6320 0 6360 4800
1024 6400 4800
```

mandelbrot_00000160 :

```
-2.3333166666666667 1.00001666666666666 -1.25 1.25 6360 0 6400 4800
1024 6400 4800
```

One Second

Each file contains an increasing integer, starting from one, representing the number of seconds the processor should wait for.

Appendix F

Experimental Results

Please see accompanying file `bcg_phd_thesis_appendix_F.pdf`